AUTOMATIZE TAREFAS MAÇANTES COM PYTHON

PROGRAMAÇÃO PRÁTICA PARA VERDADEIROS INICIANTES

AL SWEIGART

novatec

AUTOMATIZE TAREFAS MAÇANTES COM PYTHON

PROGRAMAÇÃO PRÁTICA PARA VERDADEIROS INICIANTES

Al Sweigart

Novatec

Editor: Rubens Prates
Assistente editorial: Priscila Yoshimatsu
Tradução: Lúcia A. Kinoshita
Revisão gramatical: Marta Almeida de Sá
Editoração eletrônica: Carolina Kuwabata

Para meu sobrinho Jack.

Sobre o autor

Al Sweigart é desenvolvedor de software, autor de livros técnicos e mora em San Francisco. O Python é sua linguagem de programação favorita, e o autor desenvolveu vários módulos de código aberto para essa linguagem. Seus outros livros estão disponíveis gratuitamente por meio da licença Creative Commons em seu site *http://www.inventwithpython.com/*. Seu gato pesa aproximadamente seis quilos e meio.

Sobre a revisora técnica

Ari Lacenski é desenvolvedora de aplicações Android e softwares Python. Mora em San Francisco e escreve sobre programação Android em *http://gradlewhy.ghost.io/*, além de ser orientadora do Women Who Code. Ela também toca música *folk* no violão.

SUMÁRIO

Agradecimentos

Introdução

A quem este livro se destina?
Convenções
O que é programação?
  O que é Python?
  Programadores não precisam saber muita matemática
  Programação é uma atividade criativa
Sobre este livro
Download e instalação do Python
Iniciando o IDLE
  Shell interativo
Onde encontrar ajuda
Fazendo perguntas inteligentes sobre programação
Resumo

Parte I ■ Básico da programação python

Capítulo 1 ■ Básico sobre o python

Fornecendo expressões no shell interativo
Tipos de dado inteiro, de ponto flutuante e string
Concatenação e repetição de strings
Armazenando valores em variáveis
  Instruções de atribuição
  Nomes de variáveis
Seu primeiro programa
Dissecando seu programa
  Comentários

Função print()
Função input()
Exibindo o nome do usuário
Função len()
Funções str(), int() e float()
Resumo
Exercícios práticos

Capítulo 2 ■ Controle de fluxo

Valores booleanos
Operadores de comparação
Operadores booleanos
Operadores booleanos binários
Operador not
Misturando operadores booleanos e de comparação
Elementos do controle de fluxo
Condições
Blocos de código
Execução do programa
Instrução de controle de fluxo
Instruções if
Instruções else
Instruções elif
Instruções de loop while
Instruções break
Instruções continue
Loops for e a função range()
Importando módulos
Instruções from import
Encerrando um programa previamente com sys.exit()
Resumo
Exercícios práticos

Capítulo 3 ■ Funções

Instruções def com parâmetros
Valores de retorno e instruções return
Valor None
Argumentos nomeados e print()
Escopo local e global
  Variáveis locais não podem ser usadas no escopo global
  Escopos locais não podem usar variáveis de outros escopos locais
  Variáveis globais podem ser lidas a partir de um escopo local
  Variáveis locais e globais com o mesmo nome
Instrução global
Tratamento de exceções
Um pequeno programa: adivinhe o número
Resumo
Exercícios práticos
Projetos práticos
  Sequência de Collatz
  Validação de dados de entrada

Capítulo 4 ■ Listas

Tipo de dado lista
  Obtendo valores individuais de uma lista por meio de índices
  Índices negativos
  Obtendo sublistas com slices
  Obtendo o tamanho de uma lista com len()
  Alterando valores de uma lista usando índices
  Concatenação e repetição de listas
  Removendo valores de listas usando instruções del
Trabalhando com listas
  Utilizando loops for com listas
  Operadores in e not in
  Truque da atribuição múltipla
Operadores de atribuição expandidos
Métodos
  Encontrando um valor em uma lista com o método index()

- Adicionando valores a listas com os métodos append() e insert()
- Removendo valores de listas com remove()
- Ordenando os valores de uma lista com o método sort()

Exemplo de programa: Magic 8 Ball com uma lista

Tipos semelhantes a listas: strings e tuplas
- Tipos de dados mutáveis e imutáveis
- Tipo de dado tupla
- Convertendo tipos com as funções list() e tuple()

Referências
- Passando referências
- Funções copy() e deepcopy() do módulo copy

Resumo

Exercícios práticos

Projetos práticos
- Código para vírgulas
- Grade para imagem composta de caracteres

Capítulo 5 ■ Dicionários e estruturação de dados

Tipo de dado dicionário
- Comparação entre dicionários e listas
- Métodos keys(), values() e items()
- Verificando se uma chave ou um valor estão presentes em um dicionário
- Método get()
- Método setdefault()

Apresentação elegante

Utilizando estruturas de dados para modelar objetos do mundo real
- Um tabuleiro de jogo da velha
- Dicionários e listas aninhados

Resumo

Exercícios práticos

Projetos práticos
- Inventário de um jogo de fantasia
- Função de “lista para dicionário” para o inventário de jogo de fantasia

Capítulo 6 ■ Manipulação de strings
Trabalhando com strings
Strings literais
Indexação e slicing de strings
Operadores in e not in com strings
Métodos úteis de string
Métodos de string upper(), lower(), isupper() e islower()
Métodos de string isX
Métodos de string startswith() e endswith()
Métodos de string join() e split()
Justificando texto com rjust(), ljust() e center()
Removendo espaços em branco com strip(), rstrip() e lstrip()
Copiando e colando strings com o módulo pyperclip
Projeto: Repositório de senhas
Passo 1: Design do programa e estruturas de dados
Passo 2: Tratar argumentos da linha de comando
Passo 3: Copiar a senha correta
Projeto: Adicionando marcadores na marcação da Wiki
Passo 1: Copiar e colar no clipboard
Passo 2: Separar as linhas de texto e acrescentar o asterisco
Passo 3: Juntar as linhas modificadas
Resumo
Exercícios práticos
Projeto prático
Exibição de tabela
Parte II ■ Automatizando tarefas
Capítulo 7 ■ Correspondência de padrões com expressões regulares
Encontrando padrões de texto sem usar expressões regulares
Encontrando padrões de texto com expressões regulares
Criando objetos Regex

- Objetos Regex de correspondência
- Revisão da correspondência com expressão regular

Mais correspondência de padrões com expressões regulares
- Agrupando com parênteses
- Fazendo a correspondência de vários grupos com pipe
- Correspondência opcional usando ponto de interrogação
- Correspondendo a zero ou mais ocorrências usando asterisco
- Correspondendo a uma ou mais ocorrências usando o sinal de adição
- Correspondendo a repetições específicas usando chaves

Correspondências greedy e nongreedy
Método findall()
Classes de caracteres
Criando suas próprias classes de caracteres
Acento circunflexo e o sinal de dólar
Caractere-curinga
- Correspondendo a tudo usando ponto-asterisco
- Correspondendo a quebras de linha com o caractere ponto

Revisão dos símbolos de regex
Correspondências sem diferenciar letras maiúsculas de minúsculas
Substituindo strings com o método sub()
Administrando regexes complexas
Combinando re.IGNORECASE, re.DOTALL e re.VERBOSE
Projeto: extrator de números de telefone e de endereços de email
- Passo 1: Criar uma regex para números de telefone
- Passo 2: Criar uma regex para endereços de email
- Passo 3: Encontrar todas as correspondências no texto do clipboard
- Passo 4: Reunir as correspondências em uma string para o clipboard
- Executando o programa
- Ideias para programas semelhantes

Resumo
Exercícios práticos
Projetos práticos
- Detecção de senhas robustas
- Versão de strip() usando regex

Capítulo 8 ■ Lendo e escrevendo em arquivos

Arquivos e paths de arquivo
- Barra invertida no Windows e barra para frente no OS X e no Linux
- Diretório de trabalho atual
- Comparação entre paths absolutos e relativos
- Criando novas pastas com os.makedirs()

Módulo os.path
- Lidando com paths absolutos e relativos
- Obtendo os tamanhos dos arquivos e o conteúdo das pastas
- Verificando a validade de um path

Processo de leitura/escrita
- Abrindo arquivos com a função open()
- Lendo o conteúdo dos arquivos
- Escrevendo em arquivos

Salvando variáveis com o módulo shelve

Salvando variáveis com a função pprint.pformat()

Projeto: gerando arquivos aleatórios de provas
- Passo 1: Armazenar os dados da prova em um dicionário
- Passo 2: Criar o arquivo com a prova e embaralhar a ordem das perguntas
- Passo 3: Criar as opções de resposta
- Passo 4: Gravar conteúdo nos arquivos de prova e de respostas

Projeto: Multiclipboard
- Passo 1: Comentários e configuração do shelf
- Passo 2: Salvar o conteúdo do clipboard com uma palavra-chave
- Passo 3: Listar palavras-chaves e carregar o conteúdo de uma palavra-chave

Resumo

Exercícios práticos

Projetos práticos
- Estendendo o multiclipboard
- Mad Libs
- Pesquisa com regex

Capítulo 9 ■ Organizando arquivos

Módulo shutil

Copiando arquivos e pastas
Movendo e renomeando arquivos e pastas
Apagando arquivos e pastas permanentemente
Apagando arquivos com segurança usando o módulo send2trash
Percorrendo uma árvore de diretório
Compactando arquivos com o módulo zipfile
Lendo arquivos ZIP
Extraindo itens de arquivos ZIP
Criando arquivos ZIP e adicionando itens
Projeto: Renomeando arquivos com datas em estilo americano para datas em estilo europeu
Passo 1: Criar uma regex para datas em estilo americano
Passo 2: Identificar as Partes da data nos nomes de arquivo
Passo 3: Compor o novo nome de arquivo e renomear os arquivos
Ideias para programas semelhantes
Projeto: Fazer backup de uma pasta usando um arquivo ZIP
Passo 1: Determinar o nome do arquivo ZIP
Passo 2: Criar o novo arquivo ZIP
Passo 3: Percorrer a árvore de diretório e fazer adições ao arquivo ZIP
Ideias para programas semelhantes
Resumo
Exercícios práticos
Projetos práticos
Cópia seletiva
Apagando arquivos desnecessários
Preenchendo as lacunas

Capítulo 10 ■ Debugging

Gerando exceções
Obtendo o traceback como uma string
Asserções
Usando uma asserção em uma simulação de semáforo
Desabilitando as asserções
Logging

Utilizando o módulo logging
Não faça debug com print()
Níveis de logging
Desabilitando o logging
Logging em um arquivo
Debugger do IDLE
Go
Step
Over
Out
Quit
Fazendo debugging de um programa que soma números
Breakpoints
Resumo
Exercícios práticos
Projeto prático
Debugging em um programa de lançamento de moeda

Capítulo 11 ■ Web Scraping

Projeto: mapIt.py com o módulo webbrowser
Passo 1: Identificar o URL
Passo 2: Tratar argumentos da linha de comando
Passo 3: Tratar o conteúdo do clipboard e iniciar o navegador
Ideias para programas semelhantes
Fazendo download de arquivos da Web com o módulo requests
Fazendo download de uma página web com a função requests.get()
Verificando se houve erros
Salvando arquivos baixados no disco rígido
HTML
Recursos para aprender HTML
Uma revisão rápida
Visualizando o código-fonte HTML de uma página web
Abrindo as ferramentas de desenvolvedor em seu navegador
Usando as ferramentas de desenvolvedor para encontrar elementos HTML

Fazendo parse de HTML com o módulo BeautifulSoup
Criando um objeto BeautifulSoup a partir do HTML
Encontrando um elemento com o método select()
Obtendo dados dos atributos de um elemento
Projeto: Pesquisa "Estou com sorte" no Google
Passo 1: Obter os argumentos da linha de comando e solicitar a página de pesquisa
Passo 2: Encontrar todos os resultados
Passo 3: Abrir abas do navegador web para cada resultado
Ideias para programas semelhantes
Projeto: Downloading de todas as tirinhas XKCD
Passo 1: Design do programa
Passo 2: Download da página web
Passo 3: Encontrar e fazer download da imagem da tirinha
Passo 4: Salvar a imagem e encontrar a tirinha anterior
Ideias para programas semelhantes
Controlando o navegador com o módulo Selenium
Iniciando um navegador controlado pelo Selenium
Localizando elementos na página
Clicando na página
Preenchendo e submetendo formulários
Enviando teclas especiais
Clicando nos botões do navegador
Mais informações sobre o Selenium
Resumo
Exercícios práticos
Projetos práticos
Envio de emails pela linha de comando
Programa para fazer downloads de um site de imagens
2048
Verificação de links
Capítulo 12 ■ Trabalhando com planilhas Excel
Documentos Excel

- Instalando o módulo openpyxl
- Lendo documentos Excel
  - Abrindo documentos Excel com o OpenPyXL
  - Obtendo as planilhas do workbook
  - Obtendo as células das planilhas
  - Fazendo a conversão entre letras e números das colunas
  - Obtendo linhas e colunas das planilhas
  - Workbooks, planilhas e células
- Projeto: Ler dados de uma planilha
  - Passo 1: Ler os dados da planilha
  - Passo 2: Preencher a estrutura de dados
  - Passo 3: Gravar os resultados em um arquivo
  - Ideias para programas semelhantes
- Escrevendo em documentos Excel
  - Criando e salvando documentos Excel
  - Criando e removendo planilhas
  - Escrevendo valores em células
- Projeto: Atualizando uma planilha
  - Passo 1: Definir uma estrutura de dados com as informações a serem atualizadas
  - Passo 2: Verificar todas as linhas e atualizar os preços incorretos
  - Ideias para programas semelhantes
- Definindo o estilo de fonte das células
- Objetos Font
- Fórmulas
- Ajustando linhas e colunas
  - Definido a altura da linha e a largura da coluna
  - Mesclar e separar células
  - Painéis congelados
- Gráficos
- Resumo
- Exercícios práticos
- Projetos práticos
  - Gerador de tabelas de multiplicação

Programa para inserção de linhas em branco
Programa para inverter células da planilha
Arquivos-texto para planilha
Planilhas para arquivos-texto

Capítulo 13 ■ Trabalhando com documentos PDF e Word

Documentos PDF
Extraindo texto de PDFs
Descriptografando PDFs
Criando PDFs
Projeto: Combinando páginas selecionadas de vários PDFs
Passo 1: Encontrar todos os arquivos PDF
Passo 2: Abrir cada PDF
Passo 3: Adicionar cada página
Passo 4: Salvar o resultado
Ideias para programas semelhantes
Documentos Word
Lendo documentos Word
Obtendo o texto completo de um arquivo .docx
Estilizando parágrafos e objetos Run
Criando documentos Word com estilos que não sejam default
Atributos de Run
Escrevendo em documentos Word
Adicionando títulos
Adicionando quebras de linha e de página
Adicionando imagens
Resumo
Exercícios práticos
Projetos práticos
Paranoia com PDFs
Convites personalizados como documentos Word
Programa para quebra de senha de PDF baseado em força bruta

Capítulo 14 ■ Trabalhando com arquivos CSV e dados JSON

Módulo CSV
Objetos Reader
Lendo dados de objetos Reader em um loop for
Objetos Writer
Argumentos nomeados delimiter e lineterminator
Projeto: Removendo o cabeçalho de arquivos CSV
Passo 1: Percorrer todos os arquivos CSV em um loop
Passo 2: Ler o arquivo CSV
Passo 3: Gravar o arquivo CSV sem a primeira linha
Ideias para programas semelhantes
JSON e APIs
Módulo json
Lendo JSON com a função loads()
Escrevendo JSON com a função dumps()
Projeto: Acessando dados atuais de previsão do tempo
Passo 1: Obter a localidade a partir dos argumentos da linha de comando
Passo 2: Fazer download dos dados JSON
Passo 3: Carregar dados JSON e exibir informações sobre a previsão do tempo
Ideias para programas semelhantes
Resumo
Exercícios práticos
Projeto prático
Conversor de Excel para CSV

Capítulo 15 ■ Monitorando tempo, agendando tarefas e iniciando programas

Módulo time
Função time.time()
Função time.sleep()
Arredondando números

SMTP
Enviando emails
  Conectando-se a um servidor SMTP
  Enviando a mensagem “Hello” do SMTP
  Iniciando a criptografia TLS
  Fazendo login no servidor SMTP
  Enviando um email
  Desconectando-se do servidor SMTP
IMAP
Obtendo e apagando emails com o IMAP
  Conectando-se a um servidor IMAP
  Fazendo login no servidor IMAP
  Procurando emails
  Buscando um email e marcando-o como lido
  Obtendo endereços de email de uma mensagem pura
  Obtendo o corpo de uma mensagem pura
  Apagando emails
  Desconectando-se do servidor IMAP
Projeto: Enviando emails com aviso de vencimento de pagamento
  Passo 1: Abrir o arquivo Excel
  Passo 2: Localizar todos os sócios que não fizeram o pagamento
  Passo 3: Enviar emails personalizados para servir de lembrete
Enviando mensagens de texto com o Twilio
  Criando uma conta no Twilio
  Enviando mensagens de texto
Projeto: Módulo “Envie uma mensagem a mim mesmo”
Resumo
Exercícios práticos
Projetos práticos
  Programa para enviar emails com atribuições de tarefas aleatórias
  Lembrete para pegar o guarda-chuva
  Cancelamento automático de inscrição
  Controlando seu computador por email

Capítulo 17 ■ Manipulando imagens

Básico sobre imagens no computador
  Cores e valores RGBA
  Coordenadas e tuplas de caixa
Manipulando imagens com o Pillow
  Trabalhando com o tipo de dado Image
  Recortando imagens
  Copiando e colando imagens sobre outras imagens
  Redimensionando uma imagem
  Fazendo rotações e invertendo as imagens
  Alterando pixels individuais
Projeto: Adicionando um logo
  Passo 1: Abrir a imagem com o logo
  Passo 2: Percorrer todos os arquivos e abrir as imagens em um loop
  Passo 3: Redimensionar as imagens
  Passo 4: Adicionar o logo e salvar as alterações
  Ideias para programas semelhantes
Desenhando em imagens
  Desenhando formas
  Desenhando textos
Resumo
Exercícios práticos
Projetos práticos
  Estendendo e corrigindo os programas do projeto do Capítulo
  Identificando pastas com fotos no disco rígido
  Cartões personalizados para indicar o assento

Capítulo 18 ■ Controlando o teclado e o mouse com automação de GUI

Instalando o módulo pyautogui
Permanecendo no caminho certo
  Encerrando tudo ao fazer logout
  Pausas e falhas com segurança

Controlando os movimentos do mouse
Movendo o mouse
Obtendo a posição do mouse
Projeto: “Onde está o mouse neste momento?”
Passo 1: Importar o módulo
Passo 2: Criar o código para saída e o loop infinito
Passo 3: Obter e exibir as coordenadas do mouse
Controlando a interação com o mouse
Clicando o mouse
Arrastando o mouse
Fazendo rolagens com o mouse
Trabalhando com a tela
Obtendo uma captura de tela
Analisando a tela capturada
Projeto: Estendendo o programa mouseNow
Reconhecimento de imagens
Controlando o teclado
Enviando uma string a partir do teclado
Nomes das teclas
Pressionando e soltando as teclas
Combinações para atalhos de teclado
Revisão das funções de PyAutoGUI
Projeto: Preenchimento automático de formulários
Passo 1: Identificar os passos
Passo 2: Definir as coordenadas
Step 3: Começar a digitar os dados
Passo 4: Tratar listas de seleção e botões de rádio
Passo 5: Submeter o formulário e esperar
Resumo
Exercícios práticos
Projetos práticos
Parecendo ocupado
Bot para aplicativo de mensagens instantâneas
Tutorial para bot usado em jogo

AGRADECIMENTOS

Não poderia ter escrito um livro como este sem a ajuda de várias pessoas. Gostaria de agradecer a Bill Pollock, a meus editores Laurel Chun, Leslie Shen, Greg Poulos e Jennifer Griffith-Delgado, e ao restante da equipe da No Starch Press por sua inestimável ajuda. Agradeço à minha revisora técnica Ari Lacenski pelas ótimas sugestões, correções e pelo apoio.

Muito obrigado a Guido van Rossum, nosso Benevolent Dictator For Life (Ditador benevolente vitalício), e a todos da Python Software Foundation pelo excelente trabalho. A comunidade Python é a melhor que já conheci no mercado de tecnologia.

Por fim, gostaria de agradecer à minha família, aos amigos e à turma da Shotwell’s por não se sentirem incomodados com minha vida ocupada enquanto escrevia este livro. Saúde!

INTRODUÇÃO

“Você acabou de fazer em duas horas o que nós três levamos dois dias para fazer.” Meu colega de quarto na universidade estava trabalhando em uma loja de artigos eletrônicos no início dos anos 2000. Ocasionalmente, a loja recebia uma planilha com milhares de preços de produtos de seus concorrentes. Uma equipe de três funcionários imprimia a planilha em uma pilha enorme de papel e a dividia entre si. Para cada preço de produto, eles consultavam o preço de sua loja e anotavam todos os produtos que seus concorrentes vendiam por um preço menor. Geralmente, isso exigia uns dois dias.

“Sabe, eu poderia escrever um programa que faça isso se vocês tiverem o arquivo original dos impressos”, disse meu colega de quarto a eles quando os viu sentados no chão com papéis espalhados e empilhados ao seu redor.

Depois de algumas horas, ele tinha um pequeno programa que lia o preço de um concorrente em um arquivo, localizava o produto no banco de dados da loja e anotava-o se o preço do concorrente fosse menor. Ele ainda era iniciante em programação e passou a maior parte de seu tempo consultando a documentação em um livro de programação. O programa propriamente dito exigiu somente alguns segundos para ser executado. Meu colega de quarto e seus colegas de trabalho se deram ao luxo de um almoço demorado naquele dia.

Eis a eficácia da programação de computadores. Um computador é como um canivete suíço que você pode configurar para realizar inúmeras tarefas. Muitas pessoas passam horas clicando e digitando para realizar tarefas repetitivas sem se darem conta de que o computador que estão usando poderia fazer seu trabalho em segundos se as instruções corretas fossem fornecidas.

A quem este livro se destina?

O software está no centro de várias ferramentas que usamos hoje em dia: quase todos usam redes sociais para se comunicar, muitas pessoas têm computadores conectados à Internet em seus telefones celulares e a maioria dos empregos em escritórios envolve interação com um computador para que o trabalho seja feito. Como resultado, a demanda por pessoas que saibam programar aumentou consideravelmente. Inúmeros livros, tutoriais interativos na web e treinamentos intensivos para desenvolvedores prometem transformar iniciantes ambiciosos em engenheiros de software com salários anuais de seis dígitos.

Este livro não é para essas pessoas. É para as demais.

Por si só, este livro não vai transformar você em um desenvolvedor de software profissional, do mesmo modo que algumas aulas de guitarra não converterão você em uma estrela do rock. Porém, se você é funcionário de escritório, administrador, acadêmico ou qualquer outra pessoa que utilize um computador no trabalho ou para diversão, aprenderá o básico sobre programação para poder automatizar tarefas simples como:

- mover e renomear milhares de arquivos e organizá-los em pastas;
- preencher formulários online sem que seja necessário digitar;
- fazer download de arquivos ou copiar textos de um site sempre que ele for atualizado;
- fazer seu computador enviar notificações personalizadas a você;
- atualizar ou formatar planilhas Excel;
- verificar seus emails e enviar respostas previamente redigidas.

Essas tarefas são simples, porém consomem tempo dos seres humanos e, com frequência, são tão triviais ou específicas que não há softwares prontos para realizá-las. De posse de um pouco de conhecimento de programação, você poderá fazer o seu computador realizar essas tarefas para você.

Convenções

Este livro não foi criado como um manual de referência; é um guia para iniciantes. O estilo de codificação às vezes se opõe às melhores práticas (por exemplo, alguns programas utilizam variáveis globais), porém essa é uma

contrapartida para deixar o código mais fácil de aprender. Este livro foi criado para pessoas que escrevem códigos descartáveis, e, sendo assim, não dispensamos muito tempo com estilo ou elegância. Conceitos sofisticados de programação – como programação orientada a objetos, list comprehensions (abrangência de listas) e generators (geradores) – não serão abordados por causa da complexidade que eles adicionam. Programadores veteranos podem apontar maneiras pelas quais o código deste livro poderia ser alterado para melhorar a eficiência, porém o objetivo deste livro é principalmente fazer os programas funcionarem com o menor nível possível de esforço.

O que é programação?

Os programas de TV e os filmes, com frequência, mostram programadores digitando cadeias enigmáticas de 1s e 0s furiosamente em telas brilhantes, porém a programação moderna não é tão misteriosa assim. *Programação* é simplesmente o ato de fornecer instruções para o computador executar. Essas instruções podem processar alguns números, modificar um texto, procurar informações em arquivos ou prover comunicação com outros computadores por meio da Internet.

Todos os programas utilizam instruções básicas como blocos de construção. Eis algumas das instruções mais comuns em linguagem natural:

- "Faça isso; então faça aquilo."
- "Se essa condição for verdadeira, execute essa ação; caso contrário, execute aquela ação."
- "Execute essa ação esse número de vezes."
- "Continue fazendo aquilo até essa condição ser verdadeira."

Também podemos combinar esses blocos de construção para implementar decisões mais complexas. Por exemplo, a seguir estão as instruções de programação, chamadas de *código-fonte*, para um programa simples escrito na linguagem de programação Python. Começando na parte superior, o software Python executa cada linha de código (algumas linhas serão executadas somente *se* determinada condição for verdadeira; *senão* o Python executará outra linha) até que o final seja alcançado.

```
u passwordFile = open('SecretPasswordFile.txt')
v secretPassword = passwordFile.read()
w print('Enter your password.')
  typedPassword = input()
x if typedPassword == secretPassword:
y     print('Access granted')
z     if typedPassword == '12345':
{         print('That password is one that an idiot puts on their luggage.')
  else:
|     print('Access denied')
```

Talvez você não saiba nada sobre programação, porém é provável que possa dar um palpite razoável sobre o que o código anterior faz somente ao lê-lo. Inicialmente, o arquivo *SecretPasswordFile.txt* é aberto u e a senha secreta contida nele é lida v. Em seguida, o usuário é solicitado a fornecer uma senha (por meio do teclado) w. Essas duas senhas são comparadas x e, se forem iguais, o programa exibe *Access granted* (Acesso concedido) na tela y. A seguir, o programa verifica se a senha é *12345* z e oferece uma dica informando que essa opção pode não ser a melhor para uma senha {. Se as senhas não forem iguais, o programa exibirá *Access denied* (Acesso proibido) na tela |.

O que é Python?

Python se refere à linguagem de programação (com regras de sintaxe para escrever o que é considerado um código Python válido) e ao software do interpretador Python, que lê o código-fonte (escrito na linguagem Python) e executa suas instruções. O interpretador Python é gratuito e pode ser baixado de *http://python.org/*; há versões para Linux, OS X e Windows.

O nome Python é proveniente do grupo surreal de comédia britânico Monty Python, e não do nome da cobra. Programadores Python são carinhosamente chamados de Pythonistas, e referências tanto ao Monty Python quanto a serpentes normalmente estão espalhadas pelos tutoriais e pela documentação do Python.

Programadores não precisam saber muita matemática

A ansiedade mais comum de que ouço falar sobre aprender a programar está

relacionada ao fato de as pessoas acharem que isso exige bastante matemática. Na realidade, a maior parte do que é feito em programação não exige conhecimentos matemáticos que vão além da aritmética básica. Com efeito, ser bom em programação não é tão diferente de ser bom em solucionar Sudokus.

Para resolver um Sudoku, os números de 1 a 9 devem ser preenchidos em cada linha, em cada coluna e em cada quadrado interno de 3 x 3 do quadro completo de 9 x 9. Uma solução é encontrada aplicando-se dedução e lógica a partir dos números iniciais. Por exemplo, como 5 aparece na parte superior à esquerda do Sudoku mostrado na figura 0.1, ele não poderá aparecer em nenhum outro lugar na linha superior, na coluna mais à esquerda ou no quadrado de 3 x 3 da parte superior à esquerda. Resolver uma linha, uma coluna ou um quadrado de cada vez fornecerá mais pistas numéricas para o restante do quebra-cabeça.

5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

5	3	4	6	7	8	9	1	2
6	7	2	1	9	5	3	4	8
1	9	8	3	4	2	5	6	7
8	5	9	7	6	1	4	2	3
4	2	6	8	5	3	7	9	1
7	1	3	9	2	4	8	5	6
9	6	1	5	3	7	2	8	4
2	8	7	4	1	9	6	3	5
3	4	5	2	8	6	1	7	9

Figura 0.1 – Um novo Sudoku (à esquerda) e sua solução (à direita). Apesar de usar números, o Sudoku não envolve muita matemática. (Imagens © Wikimedia Commons)

O fato de o Sudoku envolver números não quer dizer que você deva ser bom em matemática para descobrir a solução. O mesmo vale para a programação. Assim como resolver um Sudoku, escrever programas envolve dividir um problema em passos individuais e detalhados. De modo semelhante, quando fazemos *debugging* ou *depuramos* programas (ou seja, identificamos e corrigimos erros), você pode observar pacientemente o que o programa faz e identificar as causas dos bugs (erros de programação). Como todas as habilidades, quanto mais você programar, melhor você se tornará.

Programação é uma atividade criativa

A programação é uma tarefa criativa – algo como construir um castelo a partir de blocos de LEGO. Começamos com uma ideia básica de como queremos que nosso castelo se pareça e avaliamos os blocos disponíveis. Então começamos a construir. Após concluir a criação de nosso programa, podemos deixar o código mais elegante, como faríamos com nosso castelo.

A diferença entre a programação e outras atividades criativas está no fato de que, quando programamos, temos todos os materiais brutos necessários em nosso computador; não será preciso comprar nenhuma tela, tinta, filme, fios, blocos de LEGO ou componentes eletrônicos adicionais. Quando nosso programa for escrito, ele poderá ser facilmente compartilhado online com o mundo todo. Embora você possa cometer erros ao programar, a atividade continuará sendo bastante divertida.

Sobre este livro

A primeira parte deste livro inclui os conceitos básicos de programação Python; a segunda parte aborda diversas tarefas que você poderá fazer para o seu computador se automatizar. Cada capítulo da segunda parte contém programas de projetos para você estudar. Eis um breve resumo do que você encontrará em cada capítulo:

- Parte I: Básico da programação Python
  - Capítulo 1: Básico sobre o Python Inclui expressões, que são o tipo mais básico de instrução em Python, e como usar o software de shell interativo do Python para testar o código.
  - Capítulo 2: Controle de fluxo Explica como fazer os programas decidirem quais instruções devem ser executadas para que seu código possa responder de modo inteligente a diferentes condições.
  - Capítulo 3: Funções Ensina como definir suas próprias funções para que você possa organizar seu código em partes mais administráveis.
  - Capítulo 4: Listas Apresenta o tipo de dado lista e explica como organizar dados.

- Capítulo 5: Dicionários e estruturação de dados Apresenta o tipo de dado dicionário e mostra maneiras mais eficientes de organizar dados.
- Capítulo 6: Manipulação de strings Aborda o trabalho com dados do tipo texto (chamados de *strings* em Python).

- Parte II: Automatizando tarefas
  - Capítulo 7: Correspondência de padrões com expressões regulares Discute como o Python pode manipular strings e procurar padrões textuais usando expressões regulares.
  - Capítulo 8: Lendo e escrevendo em arquivos Explica como seus programas podem ler o conteúdo de arquivos do tipo texto e salvar informações em arquivos em seu disco rígido.
  - Capítulo 9: Organizando arquivos Mostra como o Python pode copiar, mover, renomear e apagar uma quantidade grande de arquivos de forma muito mais rápida do que um usuário humano poderia fazer. Também explica a compactação e a descompactação de arquivos.
  - Capítulo 10: Debugging Mostra como usar as diversas ferramentas do Python para identificação e correção de bugs.
  - Capítulo 11: Web scraping Mostra como criar programas que possam fazer download automaticamente de páginas web e fazer parse dessas páginas em busca de informações. Esse processo se chama *web scraping*.
  - Capítulo 12: Trabalhando com planilhas Excel Discute a manipulação de planilhas Excel por meio de programação para que não seja necessário lê-las. Será conveniente quando o número de documentos a ser analisado estiver na casa das centenas ou dos milhares.
  - Capítulo 13: Trabalhando com documentos PDF e Word Discute como ler documentos Word e PDF usando programação.
  - Capítulo 14: Trabalhando com arquivos CSV e dados JSON Continua explicando como manipular documentos contendo arquivos CSV e JSON usando programação.
  - Capítulo 15: Monitorando tempo, agendando tarefas e iniciando programas Explica como datas e horas são tratadas pelos programas Python e como agendar seu computador para que realize tarefas em

determinados horários. Este capítulo também mostra como seus programas Python podem iniciar programas que não tenham sido criados nessa linguagem.

- Capítulo 16: Enviando emails e mensagens de texto Explica como criar programas que possam enviar emails e mensagens de texto em seu nome.
- Capítulo 17: Manipulando imagens Explica como manipular imagens como arquivos JPEG e PNG usando programação.
- Capítulo 18: Controlando o teclado e o mouse com automação de GUI Explica como controlar o mouse e o teclado para automatizar cliques e pressionamento de teclas usando programação.

Download e instalação do Python

Você pode fazer download do Python para Windows, OS X e Ubuntu gratuitamente a partir de *http://python.org/downloads/*. Se você fizer o download da versão mais recente da página de downloads do site, todos os programas deste livro deverão funcionar.

AVISO Não se esqueça de fazer download de uma versão de Python 3 (por exemplo, 3.4.0). Os programas deste livro foram criados para executar em Python 3 e podem não funcionar corretamente – se é que vão funcionar – no Python 2.

Você encontrará arquivos de instalação do Python para computadores de 64 bits e 32 bits para cada sistema operacional na página de downloads, portanto, inicialmente, descubra qual é o arquivo de instalação de que você precisa. Se você adquiriu seu computador em 2007 ou depois disso, é mais provável que você tenha um sistema de 64 bits. Caso contrário, você terá uma versão de 32 bits, mas aqui está o modo de descobrir com certeza:

- No Windows, selecione Start4Control Panel4System (Iniciar4Painel de Controle4Sistema) e verifique se System Type (Tipo de sistema) corresponde a 64 bits ou 32 bits.
- No OS X, acesse o menu Apple, selecione About This Mac4More Info4System Report4Hardware (Sobre Este Mac4Mais informações4Relatório do Sistema4Hardware) e dê uma olhada no campo Processor Name (Nome do Processador). Se esse campo informar Intel

Core Solo ou Intel Core Duo, você tem um computador de 32 bits. Se contiver algo diferente (incluindo Intel Core 2 Duo), você tem um computador de 64 bits.

- No Ubuntu Linux, abra um Terminal e execute o comando **uname -m**. Uma resposta igual a i686 quer dizer 32 bits, enquanto x86_64 quer dizer 64 bits.

No Windows, faça download do arquivo de instalação do Python (o nome do arquivo terminará com *.msi*) e dê um clique duplo nesse arquivo. Siga as instruções apresentadas na tela para instalar o Python, conforme listadas a seguir:

1. Selecione Install for All Users (Instalar para todos os usuários) e, em seguida, clique em Next (Próximo).
2. Faça a instalação na pasta *C:\Python34* clicando em Next (Próximo).
3. Clique em Next novamente para pular a seção Customize Python (Personalizar o Python).

No Mac OS X, faça download do arquivo *.dmg* correto para a sua versão de OS X e dê um clique duplo nesse arquivo. Siga as instruções apresentadas na tela para instalar o Python, conforme listadas a seguir:

1. Quando o pacote DMG for aberto em uma nova janela, dê um clique duplo no arquivo *Python.mpkg*. Talvez seja necessário fornecer a senha de administrador.
2. Clique em Continue (Continuar) pela seção Welcome (Bem-vindo) e clique em Agree (Eu concordo) para aceitar a licença.
3. Selecione HD Macintosh (ou qualquer que seja o nome de seu disco rígido) e clique em Install (Instalar).

Se estiver executando o Ubuntu, você poderá instalar a Python a partir do Terminal seguindo estes passos:

1. Abra a janela do Terminal.
2. Digite **sudo apt-get install python3**.
3. Digite **sudo apt-get install idle3**.
4. Digite **sudo apt-get install python3-pip**.

Iniciando o IDLE

Embora o *interpretador Python* seja o software que execute seus programas Python, o *IDLE* (Interactive Development Environment, ou Ambiente de desenvolvimento interativo) é o software em que você vai digitar seus programas, de modo muito semelhante a um processador de texto. Vamos iniciar o IDLE.

- No Windows 7 ou em versões mais recentes, clique no ícone Start (Iniciar) no canto inferior esquerdo de sua tela, digite **IDLE** na caixa de pesquisa e selecione IDLE (Python GUI).
- No Windows XP, clique no botão Start (Iniciar) e selecione Programs4Python 3.44IDLE (Python GUI) [Programas4Python 3.44IDLE (Python GUI)].
- No Mac OS X, abra a janela do Finder, clique em Applications (Aplicativos), clique em Python 3.4 e, em seguida, no ícone do IDLE.
- No Ubuntu, selecione Applications4Accessories4Terminal (Aplicativos4Acessórios4Terminal) e digite **idle3**. [Você também poderá clicar em Applications (Aplicativos) na parte superior da tela, selecionar Programming (Programação) e clicar em IDLE 3.]

Shell interativo

Independentemente do sistema operacional que você estiver executando, a janela do IDLE que aparecerá inicialmente deverá estar em branco em sua maior parte, exceto pelo texto semelhante a:

```
Python 3.4.0 (v3.4.0:04f714765c13, Mar 16 2014, 19:25:23) [MSC v.1600 64 bit
(AMD64)] on win32Type "copyright", "credits" or "license()" for more information.
>>>
```

Essa janela é chamada de *shell interativo*. Um shell é um programa que permite digitar instruções ao computador de modo muito semelhante ao Terminal ou ao Command Prompt do OS X e do Windows, respectivamente. O shell interativo do Python permite fornecer instruções para serem executadas pelo software do interpretador Python. O computador lê as instruções que você fornecer e executa-as imediatamente.

Por exemplo, digite o seguinte no shell interativo, ao lado do prompt >>>:

```
>>> print('Hello world!')
```

Após ter digitado essa linha e ter teclado ENTER, o shell interativo deverá exibir a seguinte resposta:

```
>>> print('Hello world!')
Hello world!
```

Onde encontrar ajuda

Resolver problemas de programação sozinho é mais fácil do que você possa imaginar. Se não estiver convencido disso, vamos provocar intencionalmente um problema: digite **'42' + 3** no shell interativo. Não é preciso saber o que essa instrução significa neste momento, porém o resultado deverá ter a seguinte aparência:

```
  >>> '42' + 3
u Traceback (most recent call last):
    File "<pyshell#0>", line 1, in <module>
      '42' + 3
v TypeError: Can't convert 'int' object to str implicitly
  >>>
```

A mensagem de erro v apareceu aqui porque o Python não foi capaz de entender a sua instrução. A parte referente ao traceback u da mensagem de erro mostra a instrução específica e o número da linha em que o Python teve problemas. Se você não souber interpretar uma mensagem de erro em particular, faça uma pesquisa online em busca da mensagem de erro exata. Digite “TypeError: Can’t convert ‘int’ object to str implicitly” (incluindo as aspas) em sua ferramenta de pesquisa favorita e você verá inúmeros links explicando o que a mensagem de erro quer dizer e o que a causou, como mostra a figura 0.2.

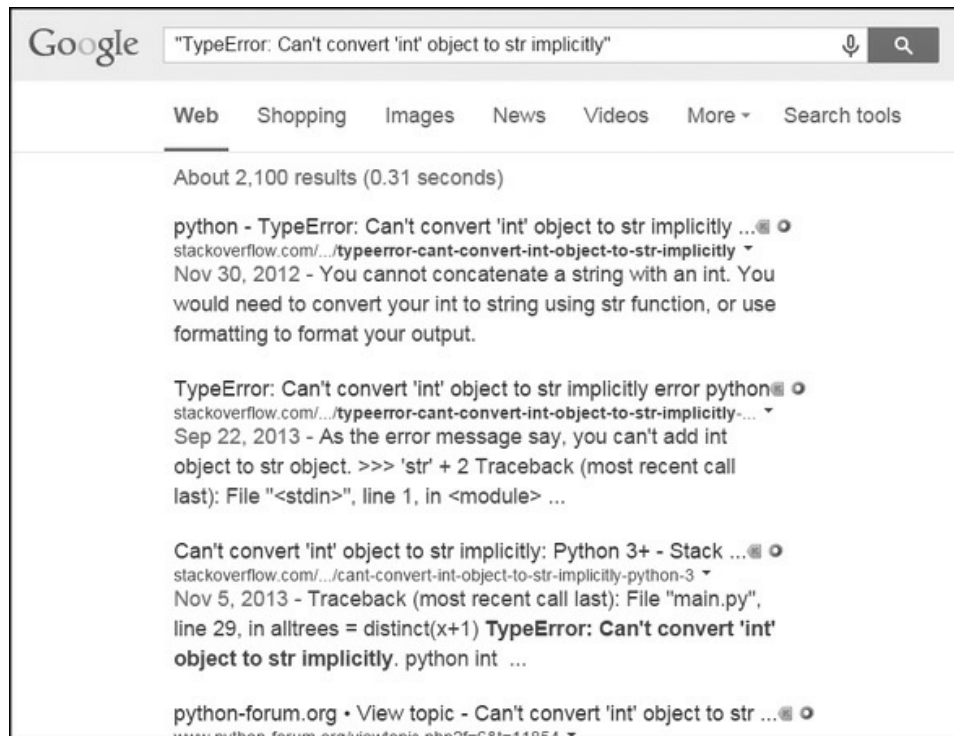



Figura 0.2 – Os resultados do Google para uma mensagem de erro podem ser muito úteis.

Com frequência, você verá que alguém já teve a mesma dúvida e que outra pessoa prestativa já respondeu. Nenhuma pessoa sozinha é capaz de saber tudo sobre programação e, sendo assim, parte do cotidiano no trabalho de qualquer desenvolvedor de software consiste em procurar respostas para perguntas técnicas.

Fazendo perguntas inteligentes sobre programação

Se você não puder encontrar a resposta fazendo pesquisas online, experimente perguntar para as pessoas em um fórum web como o Stack Overlow (*http://stackoverflow.com/*) ou no subreddit “learn programming” (aprender programação) em *http://reddit.com/r/learnprogramming/*. Contudo tenha em mente que há maneiras inteligentes de fazer perguntas sobre programação que ajudarão outras pessoas a ajudar você. Não se esqueça de ler as seções Frequently Asked Questions (Perguntas frequentes) que esses sites têm sobre a maneira apropriada de postar perguntas.

Ao elaborar perguntas sobre programação, lembre-se de fazer o seguinte:

- Explique o que você está tentando fazer em vez de apenas dizer o que você fez. Isso permite que a pessoa que vai ajudá-lo saiba se você está no caminho errado.
- Especifique o ponto em que o erro ocorre. Ele ocorre bem no início do programa ou somente depois de uma determinada ação ter sido executada?
- Copie e cole a mensagem de erro *completa* e o seu código em *http://pastebin.com/* ou em *http://gist.github.com/*.

Esses sites facilitam o compartilhamento de grandes quantidades de código com as pessoas na Web, sem o risco de perda de qualquer formatação de texto. Você pode então colocar o URL do código postado em seu email ou no post do fórum. Por exemplo, estas são algumas partes de código que postei: *http://pastebin.com/SzP2DbFx/* e *https://gist.github.com/asweigart/6912168/*.

- Explique o que você já tentou fazer para solucionar seu problema. Isso informará às pessoas que você já dedicou algum esforço para tentar resolver o problema por conta própria.
- Liste a versão do Python que você está usando. (Há algumas diferenças fundamentais entre os interpretadores Python para a versão 2 e a versão 3.) Além disso, informe o sistema operacional e a versão que você estiver executando.
- Se o erro surgiu após você ter feito uma mudança em seu código, explique exatamente o que você alterou.
- Informe se você é capaz de reproduzir o erro sempre que o programa é executado ou se ele ocorre somente depois que você realiza determinadas ações. Explique quais são essas ações se for o caso.

Sempre siga as regras da boa etiqueta online também. Por exemplo, não poste perguntas somente em letras maiúsculas nem faça exigências que não sejam razoáveis às pessoas que estiverem tentando ajudar você.

Resumo

Para a maioria das pessoas, seus computadores são apenas um equipamento, e não uma ferramenta. Porém, ao aprender a programar, você terá acesso a uma das ferramentas mais eficazes do mundo moderno, além de se divertir nesse

processo. Programar não é como realizar cirurgias no cérebro – não há problemas em amadores fazerem experimentos e cometerem erros.

Adoro ajudar as pessoas a descobrirem o Python. Escrevo tutoriais de programação em meu blog em *http://inventwithpython.com/blog/*, e você pode entrar em contato comigo enviando perguntas para *al@inventwithpython.com*.

Este livro fará você começar sem exigir nenhum conhecimento de programação, porém você poderá ter dúvidas que estarão além de seu escopo. Lembre-se de que fazer perguntas eficientes e saber onde encontrar as respostas são ferramentas de valor inestimável em sua jornada pela programação.

Vamos começar!

PARTE I
BÁSICO DA PROGRAMAÇÃO PYTHON

CAPÍTULO 1

BÁSICO SOBRE O PYTHON

A linguagem de programação Python tem uma ampla variedade de construções sintáticas, funções de biblioteca-padrão e recursos de ambiente interativo de desenvolvimento. Felizmente, você poderá ignorar a maior parte disso; você só precisará aprender o suficiente para escrever alguns pequenos programas práticos.

Entretanto você deverá aprender alguns conceitos básicos de programação antes de poder fazer algo. Como um aprendiz de feiticeiro, talvez você ache esses conceitos misteriosos e tediosos, porém, com um pouco de conhecimento e prática, você poderá comandar o seu computador como se fosse uma varinha mágica para realizar proezas incríveis.

Este capítulo tem alguns exemplos que incentivarão você a digitar no shell interativo, o que permitirá executar instruções Python, uma de cada vez, e ver os resultados instantaneamente. Usar o shell interativo é ótimo para saber o que as instruções Python básicas fazem, portanto experimente usá-lo enquanto acompanha este capítulo. Você se lembrará muito mais das tarefas que fizer do que dos textos que forem apenas lidos.

Fornecendo expressões no shell interativo

O shell interativo é executado ao iniciar o IDLE, que foi instalado com o Python na introdução. No Windows, abra o menu Start (Iniciar), selecione All Programs4Python 3.3 (Todos os programas4Python 3.3) e, em seguida, selecione IDLE (Python GUI). No OS X, selecione Applications 4MacPython 3.34IDLE (Aplicativos4MacPython 3.34IDLE). No Ubuntu, abra uma janela de Terminal e digite **idle3**.

Uma janela com o prompt >>> será apresentada: é o shell interativo. Digite **2 + 2** no prompt para fazer o Python realizar uma operação matemática simples.

```
>>> 2 + 2
```

```
4
```

A janela do IDLE agora deverá exibir um texto como:

```
Python 3.3.2 (v3.3.2:d047928ae3f6, May 16 2013, 00:06:53) [MSC v.1600 64 bit
(AMD64)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> 2 + 2
4
>>>
```

Em Python, `2 + 2` é chamado de *expressão*, que é o tipo de instrução de programação mais básico da linguagem. As expressões são constituídas de *valores* (como `2`) e de *operadores* (como `+`), e elas sempre podem ser *avaliados* como (ou seja, reduzidos a) um único valor. Isso quer dizer que podemos usar expressões em qualquer lugar no código Python em que poderíamos usar também um valor.

No exemplo anterior, `2 + 2` é avaliado como um único valor igual a `4`. Um valor único sem operadores também é considerado uma expressão, porém é avaliado somente como si mesmo, conforme mostrado a seguir:

```
>>> 2
2
```

TUDO BEM SE ERRAR!

Os programas falharão se contiverem códigos que o computador possa não entender, e isso fará o Python exibir uma mensagem de erro. Entretanto uma mensagem de erro não fará seu computador parar de funcionar, portanto não tenha medo de cometer erros. Um *crash* (falha) somente quer dizer que o programa parou de executar de forma inesperada.

Se quiser saber mais sobre uma mensagem de erro, procure o texto exato da mensagem online para descobrir mais sobre esse erro específico. Você também pode consultar os recursos em *http://nostarch.com/automatestuff/* para ver uma lista das mensagens de erro comuns em Python e seus significados.

Há vários outros operadores que também podemos usar em expressões Python. Por exemplo, a tabela 1.1 lista todos os operadores matemáticos em Python.

Tabela 1.1 – Operadores matemáticos, do maior para o menor nível de precedência

Operador	Operação	Exemplo	Avaliado como...
**	Exponencial	2 ** 3	8
%	Módulo/resto	22 % 8	6
//	Divisão inteira	22 // 8	2
/	Divisão	22 / 8	2.75
*	Multiplicação	3 * 5	15
-	Subtração	5 - 2	3
+	Adição	2 + 2	4

A ordem das operações (também chamada de *precedência*) dos operadores matemáticos em Python é semelhante àquela usada na matemática. O operador ** é avaliado em primeiro lugar; os operadores * /, // e % são avaliados a seguir, da esquerda para a direita, e os operadores + e - são avaliados por último (também da esquerda para a direita). Podemos usar parênteses para sobrepor a precedência normal caso seja necessário. Forneça as expressões a seguir no shell interativo:

```
>>> 2 + 3 * 6
20
>>> (2 + 3) * 6
30
>>> 48565878 * 578453
28093077826734
>>> 2 ** 8
256
>>> 23 / 7
3.2857142857142856
>>> 23 // 7
3
>>> 23 % 7
2
>>> 2     +          2
4
>>> (5 - 1) * ((7 + 1) / (3 - 1))
16.0
```

Em cada caso, você como programador deve fornecer a expressão, porém o

Python fará a parte difícil de avaliá-la como um único valor. O Python continuará avaliando partes da expressão até ela se transformar em um único valor, como mostra a figura 1.1.

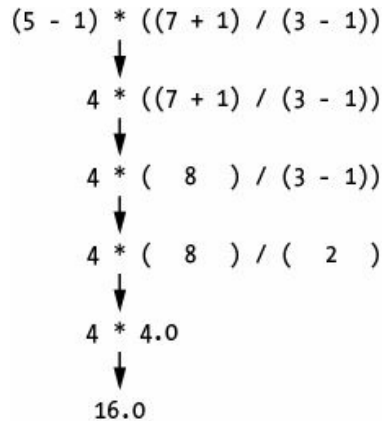



Figura 1.1 – Avaliar uma expressão a reduz a um único valor.

Essas regras para reunir operadores e valores e formar expressões são uma parte fundamental do Python como uma linguagem de programação, assim como as regras gramaticais que nos ajudam na comunicação. Aqui está um exemplo:

> Esta é uma sentença gramaticalmente correta em português.
>
> Esta gramaticalmente é sentença não em português correta uma.

A segunda linha é difícil de interpretar, pois ela não segue as regras do português. De modo semelhante, se você digitar uma instrução Python inadequada, o Python não será capaz de entendê-la e exibirá uma mensagem de erro SyntaxError, conforme mostrada a seguir:

```
>>> 5 +
  File "<stdin>", line 1
    5 +
      ^
SyntaxError: invalid syntax
>>> 42 + 5 + * 2
  File "<stdin>", line 1
    42 + 5 + * 2
             ^
SyntaxError: invalid syntax
```

É sempre possível verificar se uma instrução funciona digitando-a no shell

interativo. Não se preocupe em quebrar o computador: o pior que pode acontecer é o Python responder com uma mensagem de erro. Desenvolvedores profissionais de software obtêm mensagens de erro o tempo todo enquanto escrevem códigos.

Tipos de dado inteiro, de ponto flutuante e string

Lembre-se de que as expressões são apenas valores combinados com operadores e que elas são sempre avaliadas como um único valor. Um *tipo de dado* é uma categoria para valores, e todo valor pertence exatamente a um tipo de dado. Os tipos de dado mais comuns em Python estão listados na tabela 1.2. Os valores -2 e 30, por exemplo, são chamados de valores *inteiros* (integers). O tipo de dado inteiro (ou int) representa valores que contêm números inteiros. Os números com um ponto decimal, como 3.14, são chamados de *números de ponto flutuante* (ou floats). Observe que, apesar de o valor 42 ser um inteiro, o valor 42.0 é um número de ponto flutuante.

Tabela 1.2 – Tipos comuns de dados

Tipo de dado	Exemplos
Inteiros	-2, -1, 0, 1, 2, 3, 4, 5
Números de ponto flutuante	-1.25, -1.0, --0.5, 0.0, 0.5, 1.0, 1.25
Strings	'a', 'aa', 'aaa', 'Hello!', '11 cats'

Os programas Python também podem ter valores textuais chamados *strings* ou *strs*. Sempre insira caracteres de aspa simples (') – como em 'Hello' ou em 'Goodbye cruel world!' – ao redor de sua string para que o Python saiba em que ponto a string começa e termina. Você pode até mesmo ter uma string sem caracteres, ou seja, '', que é chamada de *string vazia*. As strings serão explicadas com mais detalhes no capítulo 4.

Se você vir a mensagem de erro SyntaxError: EOL while scanning string literal (SyntaxError: EOL enquanto analisava a string literal), é provável que tenha esquecido o último caractere de aspas simples no final da string, como no exemplo a seguir:

```
>>> 'Hello world!
SyntaxError: EOL while scanning string literal
```

Concatenação e repetição de strings

O significado de um operador pode mudar de acordo com os tipos de dados dos valores próximos a ele. Por exemplo, + é o operador de adição quando atua sobre dois valores inteiros ou de ponto flutuante. Entretanto, quando + é usado com dois valores do tipo string, ele une as strings, constituindo o operador de *concatenação de strings*. Digite o seguinte no shell interativo:

```
>>> 'Alice' + 'Bob'
'AliceBob'
```

A expressão é avaliada como um único valor novo do tipo string que combina o texto das duas strings. Entretanto, se você tentar usar o operador + em uma string e um valor inteiro, o Python não saberá como lidar com isso e exibirá uma mensagem de erro.

```
>>> 'Alice' + 42
Traceback (most recent call last):
  File "<pyshell#26>", line 1, in <module>
    'Alice' + 42
TypeError: Can't convert 'int' object to str implicitly
```

A mensagem de erro Can't convert 'int' object to str implicitly (Não foi possível converter o objeto ‘int’ para str implicitamente) quer dizer que o Python achou que você estava tentando concatenar um inteiro à string 'Alice'. Seu código deverá converter explicitamente o inteiro em uma string, pois o Python não pode fazer isso automaticamente. (A conversão de tipos de dados será explicada na seção “Dissecando seu programa”, em que as funções str(), int() e float() serão discutidas.)

O operador * é usado para multiplicação quando atua sobre dois valores inteiros ou de ponto flutuante. Porém, quando for usado em um valor do tipo string e um valor inteiro, o operador * corresponderá ao operador de *repetição de string*. Forneça uma string multiplicada por um número no shell interativo para ver esse operador em ação:

```
>>> 'Alice' * 5
'AliceAliceAliceAliceAlice'
```

A expressão é avaliada como um único valor do tipo string que repete o

valor original um número de vezes igual ao valor inteiro. A repetição de string é um truque útil, porém não é usada com tanta frequência quanto a concatenação de strings.

O operador * pode ser usado apenas com dois valores numéricos (para multiplicação) ou com um valor do tipo string e um valor inteiro (para repetição de string). Caso contrário, o Python simplesmente exibirá uma mensagem de erro.

```
>>> 'Alice' * 'Bob'
Traceback (most recent call last):
  File "<pyshell#32>", line 1, in <module>
    'Alice' * 'Bob'
TypeError: can't multiply sequence by non-int of type 'str'
>>> 'Alice' * 5.0
Traceback (most recent call last):
  File "<pyshell#33>", line 1, in <module>
    'Alice' * 5.0
TypeError: can't multiply sequence by non-int of type 'float'
```

O fato de o Python não entender essas expressões faz sentido: não podemos multiplicar duas palavras, e é difícil repetir uma string qualquer um número fracionário de vezes.

Armazenando valores em variáveis

Uma *variável* é como uma caixa na memória do computador, em que podemos armazenar um único valor. Se quiser usar posteriormente o resultado de uma expressão avaliada em seu programa, você poderá salvá-la em uma variável.

Instruções de atribuição

Valores são armazenados em variáveis por meio de uma *instrução de atribuição*. Uma instrução de atribuição consiste de um nome de variável, um sinal de igualdade (chamado de *operador de atribuição*) e o valor a ser armazenado. Se a instrução de atribuição spam = 42 for especificada, a variável chamada spam conterá o valor inteiro 42.

Pense em uma variável como uma caixa etiquetada em que um valor é

inserido, como mostra a figura 1.2.

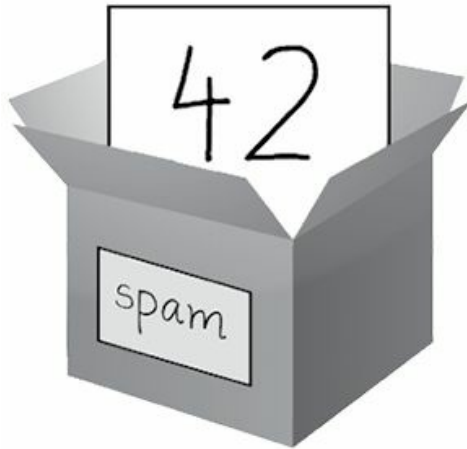



Figura 1.2 – É como se spam = 42 *dissesse ao programa que "a variável* spam *agora contém o valor inteiro* 42*".*

Por exemplo, digite o seguinte no shell interativo:

```
u >>> spam = 40
  >>> spam
  40
  >>> eggs = 2
v >>> spam + eggs
  42
  >>> spam + eggs + spam
  82
w >>> spam = spam + 2
  >>> spam
  42
```

Uma variável é *inicializada* (ou criada) na primeira vez que um valor é armazenado nela u. Depois disso, você poderá usá-la em expressões, juntamente com outras variáveis e outros valores v. Quando uma variável recebe um novo valor w, o valor antigo é esquecido, motivo pelo qual spam é avaliado com 42, e não com 40, no final do exemplo. Esse processo se chama *sobrescrever* a variável. Digite o código a seguir no shell interativo para tentar sobrescrever uma string:

```
>>> spam = 'Hello'
>>> spam
'Hello'
```

```
>>> spam = 'Goodbye'
>>> spam
'Goodbye'
```

Assim como a caixa da figura 1.3, a variável spam nesse exemplo armazena 'Hello' até você substituí-la por 'Goodbye'.

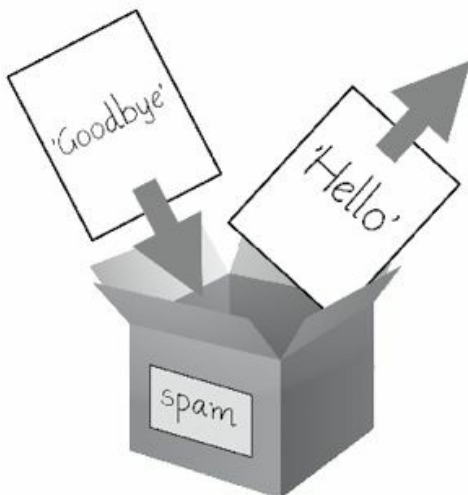



Figura 1.3 – Quando um novo valor é atribuído a uma variável, o valor antigo é esquecido.

Nomes de variáveis

A tabela 1.3 apresenta exemplos de nomes permitidos para variáveis. Você pode dar qualquer nome a uma variável desde que ele obedeça às três regras a seguir:

1. O nome pode ser constituído somente de uma palavra.
2. Somente letras, números e o caractere underscore (_) podem ser usados.
3. O nome não pode começar com um número.

Há distinção entre letras maiúsculas e minúsculas nos nomes de variáveis (são case sensitive); isso quer dizer que spam, SPAM, Spam e sPaM são quatro variáveis diferentes. Iniciar as variáveis com uma letra minúscula é uma convenção do Python.

Tabela 1.3 – Nomes válidos e inválidos de variáveis

Nomes válidos de variáveis	Nomes inválidos de variáveis
balance	current-balance (hifens não são permitidos)

currentBalance	current balance (espaços não são permitidos)
current_balance	4account (não pode começar com um número)
_spam	42 (não pode começar com um número)
SPAM	total_$um (caracteres especiais como $ não são permitidos)
account4	'hello' (caracteres especiais como ' não são permitidos)

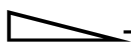
Este livro utiliza camelcase para os nomes das variáveis no lugar de underscores, ou seja, usa variáveis como lookLikeThis em vez de looking_like_this. Alguns programadores experientes podem enfatizar que o estilo oficial de codificação Python, a PEP 8, afirma que underscores devem ser usados. Sem fazer apologia, prefiro usar camelcase e destaco “A Foolish Consistency Is the Hobgoblin of Little Minds” (Uma consistência tola é o demônio das mentes medíocres) na própria PEP 8:

> *“A consistência usando o guia de estilos é importante. Porém, acima de tudo, saiba quando ser inconsistente – às vezes, o guia de estilo simplesmente não se aplica. Na dúvida, utilize o bom senso.”*[1]

Um bom nome de variável descreve os dados que ela contém. Suponha que você tenha se mudado para uma casa nova e tenha etiquetado todas as caixas de mudança como *Objetos*. Você jamais encontrará nada! Os nomes de variáveis spam, eggs e bacon são usados como nomes genéricos nos exemplos deste livro e em boa parte da documentação do Python (inspirados no esquete “Spam” do Monty Python), porém, em seus programas, um nome descritivo ajudará a deixar seu código mais legível.

Seu primeiro programa

Embora o shell interativo seja conveniente para executar uma instrução Python de cada vez, para criar programas Python completos, você deverá digitar as instruções no editor de arquivo. O *editor de arquivo* é semelhante aos editores de texto como o Notepad ou o TextMate, porém tem algumas funcionalidades específicas para a digitação de código-fonte. Para abrir o editor de arquivo no IDLE, selecione File4New Window (Arquivo4Nova janela).

A janela apresentada deverá conter um cursor à espera de seus dados de entrada, porém será diferente do shell interativo, que executa as instruções Python assim que você tecla ENTER. O editor de arquivo permite digitar diversas instruções, salvar o arquivo e executar o programa. Eis o modo de diferenciar o editor de arquivo do shell interativo:

• A janela do shell interativo sempre será aquela que contém o prompt >>>.

• A janela do editor de arquivo não contém o prompt >>>.

Agora é hora de criar o seu primeiro programa! Quando a janela do editor de arquivo abrir, digite o código a seguir:

```
u # Este programa diz hello e pergunta o meu nome.

v print('Hello world!')
  print('What is your name?')    # pergunta o nome
w myName = input()
x print('It is good to meet you, ' + myName)
y print('The length of your name is:')
  print(len(myName))
z print('What is your age?')    # pergunta a idade
  myAge = input()
  print('You will be ' + str(int(myAge) + 1) + ' in a year.')
```

Após ter inserido o código-fonte, salve-o para que não seja necessário digitá-lo novamente sempre que o IDLE for iniciado. No menu que está na parte superior da janela do editor de arquivo, selecione File4Save As (Arquivo4Salvar como). Na janela Save As (Salvar como), digite **hello.py** no campo File Name (Nome do arquivo) e, em seguida, clique em Save (Salvar).

Salve seus programas de vez em quando enquanto estiver digitando-os. Dessa maneira, se o computador falhar ou você sair acidentalmente do IDLE, o código não será perdido. Como atalho, você pode teclar CTRL-S no Windows e no Linux ou -S no OS X para salvar seu arquivo.

Após ter salvado o programa, vamos executá-lo. Selecione Run4Run Module (Executar4Executar módulo) ou simplesmente pressione a tecla F5. Seu programa deverá executar na janela do shell interativo que apareceu quando o IDLE foi iniciado. Lembre-se de que você deve pressionar F5 na janela do editor de arquivo, e não na janela do shell interativo. Forneça o seu

nome quando o programa solicitar. A saída do programa no shell interativo deverá ter a seguinte aparência:

```
Python 3.3.2 (v3.3.2:d047928ae3f6, May 16 2013, 00:06:53) [MSC v.1600 64 bit
(AMD64)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> ================================ RESTART
================================
>>>
Hello world!
What is your name?
Al
It is good to meet you, Al
The length of your name is:
2
What is your age?
4
You will be 5 in a year.
>>>
```

Quando não houver mais linhas de código para executar, o programa Python *encerrará*, ou seja, deixará de executar. (Também podemos dizer que o programa Python *saiu*.)

Você pode fechar o editor de arquivo clicando no X na parte superior da janela. Para recarregar um programa salvo, selecione File4Open (Arquivo4Abrir) no menu. Faça isso agora; na janela que aparecer, selecione *hello.py* e clique no botão Open (Abrir). Seu programa *hello.py* salvo anteriormente deverá ser aberto na janela do editor de arquivo.

Dissecando seu programa

Com seu novo programa aberto no editor de arquivo, vamos fazer um tour rápido pelas instruções Python utilizadas observando o que cada linha de código faz.

Comentários

A linha a seguir é chamada de *comentário*.

```
u # Este programa diz hello e pergunta o meu nome.
```

O Python ignora comentários e você pode usá-los para escrever notas ou para que você mesmo se lembre do que o código está tentando fazer. Qualquer texto até o final da linha após o sinal de sustenido (#) faz parte de um comentário.

Às vezes, os programadores colocarão um # na frente de uma linha de código para removê-la temporariamente enquanto estiverem testando um programa. Isso se chama *comentar* o código e pode ser útil quando você estiver tentando descobrir por que um programa não funciona. O # poderá ser removido posteriormente, quando você estiver pronto para colocar a linha de volta.

O Python também ignora a linha em branco após o comentário. Você pode acrescentar quantas linhas em branco quiser em seu programa. Isso pode deixar o seu código mais fácil de ler, como se fossem parágrafos em um livro.

Função print()

A função print() exibe na tela o valor do tipo string que está entre parênteses.

```
v print('Hello world!')
  print('What is your name?') # pergunta o nome
```

A linha print('Hello world!') quer dizer "exiba o texto que está na string 'Hello world!'". Quando o Python executa essa linha, dizemos que ele está *chamando* a função print() e que o valor do tipo string está sendo *passado* para a função. Um valor passado para uma função chama-se *argumento*. Observe que as aspas não são exibidas na tela. Elas simplesmente marcam os locais em que a string começa e termina e não fazem parte do valor da string.

NOTA Também podemos usar essa função para inserir uma linha em branco na tela; basta chamar **print()** sem nada entre parênteses.

Ao escrever um nome de função, os parênteses de abertura e de fechamento no final o identificam como o nome de uma função. É por isso que, neste livro, você verá print() em vez de print. O capítulo 2 descreve as funções com mais detalhes.

Função input()

A função input() espera o usuário digitar um texto no teclado e pressionar ENTER.

```
w myName = input()
```

Essa chamada de função é avaliada como uma string igual ao texto do usuário, e a linha de código anterior atribui o valor dessa string à variável myName.

Você pode pensar na chamada da função input() como uma expressão avaliada com qualquer string que o usuário digitar. Se o usuário digitar 'Al', a expressão será avaliada como myName = 'Al'.

Exibindo o nome do usuário

A chamada a seguir a print() contém a expressão 'It is good to meet you, ' + myName entre parênteses.

```
x print('It is good to meet you, ' + myName)
```

Lembre-se de que as expressões sempre podem ser avaliadas como um único valor. Se 'Al' for o valor armazenado em myName na linha anterior, essa expressão será avaliada como 'It is good to meet you, Al'. Esse valor único de string então é passado para print(), que o exibirá na tela.

Função len()

Podemos passar um valor de string à função len() (ou uma variável contendo uma string), e a função será avaliada como o valor inteiro referente à quantidade de caracteres dessa string.

```
y print('The length of your name is:')
  print(len(myName))
```

Digite o seguinte no shell interativo para testar isso:

```
>>> len('hello')
5
>>> len('My very energetic monster just scarfed nachos.')
46
```

```
>>> len('')
0
```

Assim como nesses exemplos, len(myName) é avaliado como um inteiro. Esse valor é então passado para print() para ser exibido na tela. Observe que print() permite que tanto valores inteiros quanto strings sejam passados a ele. Contudo observe o erro mostrado quando digitamos o seguinte no shell interativo:

```
>>> print('I am ' + 29 + ' years old.')
Traceback (most recent call last):
  File "<pyshell#6>", line 1, in <module>
    print('I am ' + 29 + ' years old.')
TypeError: Can't convert 'int' object to str implicitly
```

Não é a função print() que está causando esse erro, mas a expressão que você tentou passar para ela. Você obterá a mesma mensagem de erro se digitar a expressão sozinha no shell interativo.

```
>>> 'I am ' + 29 + ' years old.'
Traceback (most recent call last):
  File "<pyshell#7>", line 1, in <module>
    'I am ' + 29 + ' years old.'
TypeError: Can't convert 'int' object to str implicitly
```

O Python mostra um erro porque podemos usar o operador + somente para somar dois inteiros ou concatenar duas strings. Não podemos somar um inteiro e uma string, pois isso é agramatical em Python. Isso pode ser corrigido usando uma versão em string do inteiro, conforme explicado na próxima seção.

Funções str(), int() e float()

Se quiser concatenar um inteiro como 29 a uma string e passar o resultado para print(), será necessário obter o valor '29', que é forma em string de 29. A função str() pode receber um valor inteiro e será avaliada como uma versão em string desse valor, da seguinte maneira:

```
>>> str(29)
'29'
```

```
>>> print('I am ' + str(29) + ' years old.')
I am 29 years old.
```

Como str(29) é avaliado como '29', a expressão 'I am ' + str(29) + ' years old.' será avaliada como 'I am ' + '29' + ' years old.' que, por sua vez, será avaliada como 'I am 29 years old.'. Esse é o valor passado para a função print().

As funções str(), int() e float() serão respectivamente avaliadas como as formas em string, inteiro e de ponto flutuante do valor que você passar. Experimente converter alguns valores no shell interativo usando essas funções e observe o que acontece.

```
>>> str(0)
'0'
>>> str(-3.14)
'-3.14'
>>> int('42')
42
>>> int('-99')
-99
>>> int(1.25)
1
>>> int(1.99)
1
>>> float('3.14')
3.14
>>> float(10)
10.0
```

Os exemplos anteriores chamam as funções str(), int() e float() e passam valores de outros tipos de dados para obter a forma em string, inteiro ou de ponto flutuante desses valores.

A função str() será conveniente quando houver um inteiro ou um número de ponto flutuante que você queira concatenar em uma string. A função int() também será útil se houver um número na forma de string que você queira usar em alguma operação matemática. Por exemplo, a função input() sempre retorna uma string, mesmo que o usuário tenha fornecido um número. Digite **spam = input()** no shell interativo e forneça **101** quando seu texto estiver sendo esperado.

```
>>> spam = input()
101
>>> spam
'101'
```

O valor armazenado em spam não é o inteiro 101, mas a string '101'. Se quiser realizar alguma operação matemática usando o valor em spam, utilize a função int() para obter a forma inteira de spam e, em seguida, armazene esse resultado como o novo valor de spam.

```
>>> spam = int(spam)
>>> spam
101
```

Agora você poderá tratar a variável spam como um inteiro em vez de tratá-la como string.

```
>>> spam * 10 / 5
202.0
```

Observe que, se você passar um valor para int() que não possa ser avaliado como inteiro, o Python exibirá uma mensagem de erro.

```
>>> int('99.99')
Traceback (most recent call last):
  File "<pyshell#18>", line 1, in <module>
    int('99.99')
ValueError: invalid literal for int() with base 10: '99.99'
>>> int('twelve')
Traceback (most recent call last):
  File "<pyshell#19>", line 1, in <module>
    int('twelve')
ValueError: invalid literal for int() with base 10: 'twelve'
```

A função int() também será útil se você precisar arredondar um número de ponto flutuante para baixo.

```
>>> int(7.7)
7
>>> int(7.7) + 1
8
```

Em seu programa, você utilizou as funções int() e str() nas três últimas linhas

para obter um valor com o tipo de dado apropriado ao código.

```
z print('What is your age?') # pergunta a idade
  myAge = input()
  print('You will be ' + str(int(myAge) + 1) + ' in a year.')
```

A variável myAge contém o valor retornado por input(). Como a função input() sempre retorna uma string (mesmo que o usuário tenha digitado um número), podemos usar o código int(myAge) para retornar um valor inteiro a partir da string em myAge. Esse valor inteiro é então somado a 1 na expressão int(myAge) + 1.

O resultado dessa adição é passado para a função str(): str(int(myAge) + 1). O valor em string retornado é concatenado às strings 'You will be ' e ' in a year.' para ser avaliado como um valor mais extenso de string. Essa string maior é finalmente passada para print() para ser exibida na tela.

Vamos supor que o usuário forneça a string '4' para myAge. A string '4' é convertida em um inteiro para que você possa somar um a esse valor. O resultado é 5. A função str() converte o resultado de volta em uma string para que ela possa ser concatenada à segunda string 'in a year.' e a mensagem final seja criada. Esses passos de avaliação são semelhantes ao que está sendo mostrado na figura 1.4.

EQUIVALÊNCIA ENTRE TEXTO E NÚMERO

Embora o valor em string de um número seja considerado um valor totalmente diferente da versão inteira ou de ponto flutuante, um inteiro pode ser igual a um número de ponto flutuante.

```
>>> 42 == '42'
False
>>> 42 == 42.0
True
>>> 42.0 == 0042.000
True
```

O Python faz essa distinção porque as strings são texto, enquanto tanto inteiros quanto números de ponto flutuante são números.

```
print('You will be ' + str(int(myAge) + 1) + ' in a year.')

print('You will be ' + str(int( '4' ) + 1) + ' in a year.')

print('You will be ' + str(    4 + 1    ) + ' in a year.')

print('You will be ' + str(      5      ) + ' in a year.')

print('You will be ' +         '5'        + ' in a year.')

print('You will be 5'                     + ' in a year.')

print('You will be 5 in a year.')
```

Figura 1.4 – Os passos da avaliação se 4 *estiver armazenado em* myAge*.*

Resumo

Podemos calcular o valor de expressões usando uma calculadora ou concatenar strings em um processador de texto. Podemos até mesmo repetir strings facilmente copiando e colando o texto. Porém as expressões e os valores de seus componentes – operadores, variáveis e chamadas de função – constituem os blocos de construção básicos que compõem os programas. Depois que souber lidar com esses elementos, você poderá instruir o Python a realizar operações em grandes quantidades de dados.

É bom lembrar-se dos diferentes tipos de operadores (+, -, *, /, //, % e ** para operações matemáticas e + e * para operações com strings) e dos três tipos de dados (inteiros, números de ponto flutuante e strings) apresentados neste capítulo.

Algumas funções diferentes também foram apresentadas. As funções print() e input() tratam saída (na tela) e entrada (do teclado) de textos simples. A função len() recebe uma string e é avaliada como um inteiro correspondente ao número de caracteres da string. As funções str(), int() e float() serão avaliadas como as formas em string, inteiro e de ponto flutuante do valor que receberem.

No próximo capítulo, aprenderemos a dizer ao Python para tomar decisões inteligentes sobre o código a ser executado, quais códigos devem ser ignorados e quais devem ser repetidos de acordo com os valores presentes no

código. Isso é conhecido como *controle de fluxo* e permite escrever programas que tomem decisões inteligentes.

Exercícios práticos

1. Quais das opções a seguir são operadores e quais são valores?

```
*
'hello'
-88.8
-
/
+
5
```

2. Qual das opções a seguir é uma variável e qual é uma string?

```
spam
'spam'
```

3. Nomeie três tipos de dados.
4. De que é composta uma expressão? O que fazem as expressões?
5. Este capítulo apresentou as instruções de atribuição, por exemplo, `spam = 10`. Qual é a diferença entre uma expressão e uma instrução?
6. O que a variável `bacon` conterá após o código a seguir ser executado?

```
bacon = 20
bacon + 1
```

7. Para que valores as duas expressões a seguir serão avaliadas?

```
'spam' + 'spamspam'
'spam' * 3
```

8. Por que `eggs` é um nome válido de variável enquanto `100` é inválido?
9. Quais três funções podem ser usadas para obter uma versão inteira, de ponto flutuante ou em string de um valor?
10. Por que a expressão a seguir causa um erro? Como você pode corrigi-la?

```
'I have eaten ' + 99 + ' burritos.'
```

Pontos extras: pesquise a documentação do Python online em busca da função len(). Ela está em uma página web chamada “Built-in Functions” (Funções internas). Passe os olhos pela lista das demais funções Python, verifique o que a função round() faz e realize experimentos com ela no shell interativo.

---

1 N.T.: Tradução livre de acordo com a citação original em inglês: “Consistency with the style guide is important. But most importantly: know when to be inconsistent – sometimes the style guide just doesn’t apply. When in doubt, use your best judgment.”

CAPÍTULO 2

CONTROLE DE FLUXO

Você conhece o básico sobre instruções individuais e sabe que um programa é somente uma série de instruções. Contudo a verdadeira eficácia da programação não está somente em executar uma instrução após a outra, como se fosse uma lista de tarefas para o final de semana. De acordo com o modo como as expressões são avaliadas, o programa poderá decidir pular instruções, repeti-las ou escolher uma entre várias

instruções para executar. Com efeito, raramente vamos querer que nossos programas iniciem na primeira linha de código e simplesmente executem todas as linhas diretamente até o final. As *instruções de controle de fluxo* podem decidir quais instruções Python devem ser executadas de acordo com determinadas condições.

Essas instruções de controle de fluxo correspondem diretamente aos símbolos de um fluxograma; sendo assim, fornecerei versões na forma de fluxograma para os códigos discutidos neste capítulo. A figura 2.1 mostra um fluxograma para o que deve ser feito se estiver chovendo. Siga o caminho composto pelas setas do Início ao Fim.

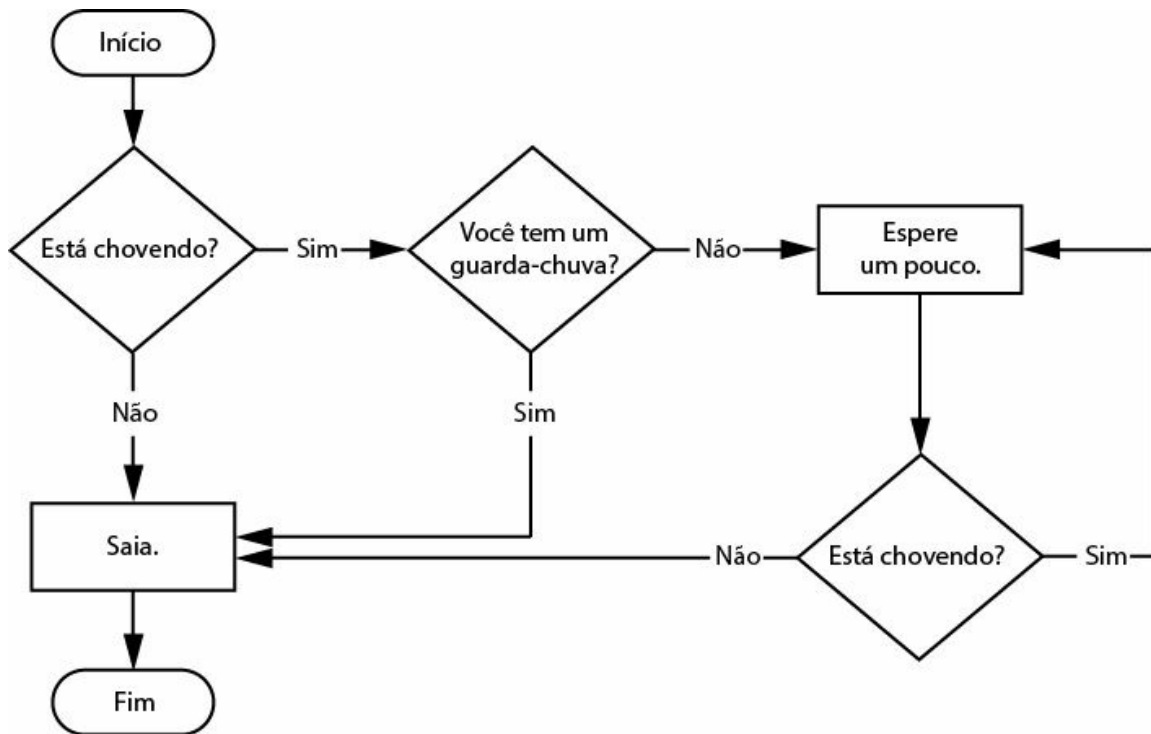



Figura 2.1 – Um fluxograma que informa o que você deve fazer se estiver chovendo.

Em um fluxograma, geralmente há mais de uma maneira de ir do início ao fim. O mesmo vale para as linhas de código em um programa de computador. Os fluxogramas representam esses pontos de ramificação com losangos, enquanto os demais passos são representados por retângulos. Os passos inicial e final são representados por retângulos com cantos arredondados.

Contudo, antes de conhecer as instruções de controle de fluxo, inicialmente você deve aprender a representar essas opções *sim* e *não*, além de saber como escrever esses pontos de ramificação na forma de código Python. Para isso, vamos explorar os valores booleanos, os operadores de comparação e os operadores booleanos.

Valores booleanos

Enquanto os tipos de dados inteiro, de ponto flutuante e string têm um número ilimitado de valores possíveis, o tipo de dado *booleano* (Boolean) tem apenas dois valores: True e False. (O booleano é um tipo de dado que recebeu seu nome em homenagem ao matemático George Boole.) Quando digitado como código Python, os valores booleanos True e False não têm aspas

em torno como as strings e sempre começam com uma letra *T* ou *F* maiúscula, com o restante da palavra em letras minúsculas. Digite o seguinte no shell interativo (algumas das instruções a seguir estão intencionalmente incorretas e farão surgir mensagens de erro):

```
u >>> spam = True
  >>> spam
  True
v >>> true
  Traceback (most recent call last):
    File "<pyshell#2>", line 1, in <module>
      true
  NameError: name 'true' is not defined
w >>> True = 2 + 2
  SyntaxError: assignment to keyword
```

Como qualquer outro valor, os valores booleanos são usados em expressões e podem ser armazenados em variáveis u. Se você não utilizar apropriadamente as letras maiúsculas e minúsculas v ou se tentar usar True e False como nomes de variáveis w, o Python apresentará uma mensagem de erro.

Operadores de comparação

Os *operadores de comparação* comparam dois valores e são avaliados como um único valor booleano. A tabela 2.1 lista os operadores de comparação.

Tabela 2.1 – Operadores de comparação

Operador	Significado
==	Igual a
!=	Diferente de
<	Menor que
>	Maior que
<=	Menor ou igual a
>=	Maior ou igual a

Esses operadores são avaliados como True ou False de acordo com os valores que você lhes fornecer. Vamos testar alguns operadores agora, começando

com == e !=.

```
>>> 42 == 42
True
>>> 42 == 99
False
>>> 2 != 3
True
>>> 2 != 2
False
```

Conforme esperado, == (igual a) é avaliado como True quando os valores em ambos os lados são iguais e != (diferente de) é avaliado como True quando os dois valores são diferentes. Os operadores == e !=, na verdade, podem funcionar com valores de qualquer tipo de dado.

```
>>> 'hello' == 'hello'
True
>>> 'hello' == 'Hello'
False
>>> 'dog' != 'cat'
True
>>> True == True
True
>>> True != False
True
>>> 42 == 42.0
True
u >>> 42 == '42'
False
```

Observe que um valor inteiro ou de ponto flutuante sempre será diferente de um valor do tipo string. A expressão 42 == '42' u é avaliada como False, pois o Python considera o inteiro 42 diferente da string '42'.

Os operadores <, >, <= e >=, por outro lado, funcionam apropriadamente somente com valores inteiros e de ponto flutuante.

```
>>> 42 < 100
True
>>> 42 > 100
False
```

```
>>> 42 < 42
False
>>> eggCount = 42
u >>> eggCount <= 42
True
>>> myAge = 29
v >>> myAge >= 10
True
```

A DIFERENÇA ENTRE OS OPERADORES == E =

Você deve ter percebido que o operador == (igual a) tem dois sinais de igualdade, enquanto o operador = (atribuição) tem apenas um. É fácil confundir esses dois operadores. Basta se lembrar dos pontos a seguir:

- O operador == (igual a) pergunta se dois valores são iguais.
- O operador = (atribuição) coloca o valor à direita na variável da esquerda.

Para ajudar a lembrar qual é qual, observe que o operador == (igual a) é constituído de dois caracteres, assim como o operador != (diferente de) também.

Com frequência, você usará operadores de comparação para comparar o valor de uma variável com outro valor, como nos exemplos eggCount <= 42 u e myAge >= 10 v. (Afinal de contas, em vez de digitar **'dog' != 'cat'** em seu código, você poderia ter simplesmente digitado **True**.) Veremos mais exemplos disso posteriormente, quando conhecermos as instruções de controle de fluxo.

Operadores booleanos

Os três operadores booleanos (and, or e not) são usados para comparar valores booleanos. Assim como os operadores de comparação, eles avaliam essas expressões reduzindo-as a um valor booleano. Vamos explorar esses operadores em detalhes, começando pelo operador and.

Operadores booleanos binários

Os operadores and e or sempre aceitam dois valores booleanos (ou expressões), portanto são considerados operadores *binários*. O operador and avalia uma expressão como True se *ambos* os valores booleanos forem True;

caso contrário, ela será avaliada como False. Forneça algumas expressões usando and no shell interativo para vê-lo em ação.

```
>>> True and True
True
>>> True and False
False
```

Uma *tabela verdade* mostra todos os resultados possíveis de um operador booleano. A tabela 2.2 contém a tabela verdade do operador and.

Tabela 2.2 – A tabela verdade do operador and

Expressão	Avaliada como...
True and True	True
True and False	False
False and True	False
False and False	False

Por outro lado, o operador or avalia uma expressão como True se *um dos* valores booleanos for True. Se ambos forem False, ela será avaliada como False.

```
>>> False or True
True
>>> False or False
False
```

Podemos ver todos os resultados possíveis do operador or em sua tabela verdade, mostrada na tabela 2.3.

Tabela 2.3 – A tabela verdade do operador or

Expressão	Avaliada como...
True or True	True
True or False	True
False or True	True
False or False	False

Operador not

De modo diferente de and e or, o operador not atua somente sobre um valor

booleano (ou uma expressão). O operador not simplesmente é avaliado como o valor booleano oposto.

```
  >>> not True
  False
u >>> not not not not True
  True
```

De modo muito semelhante ao uso de dupla negação na fala e na escrita, podemos aninhar operadores not u, embora não haja nenhum motivo para fazer isso em programas de verdade. A tabela 2.4 mostra a tabela verdade de not.

Tabela 2.4 – A tabela verdade do operador not

Expressão	Avaliada como...
not True	False
not False	True

Misturando operadores booleanos e de comparação

Como os operadores de comparação são avaliados como valores booleanos, podemos usá-los em expressões com operadores booleanos.

Lembre-se de que os operadores and, or e not são chamados de operadores booleanos porque sempre atuam sobre os valores booleanos True e False. Embora expressões como 4 < 5 não sejam valores booleanos, elas são expressões avaliadas como valores booleanos. Experimente fornecer algumas expressões booleanas que utilizem operadores de comparação no shell interativo.

```
>>> (4 < 5) and (5 < 6)
True
>>> (4 < 5) and (9 < 6)
False
>>> (1 == 2) or (2 == 2)
True
```

O computador avaliará inicialmente a expressão à esquerda e, em seguida, avaliará a expressão à direita. Quando souber o valor booleano de cada lado,

ele avaliará então a expressão toda, reduzindo-a a um único valor booleano. Podemos pensar no processo de avaliação do computador para (4 < 5) and (5 < 6) conforme mostrado na figura 2.2.

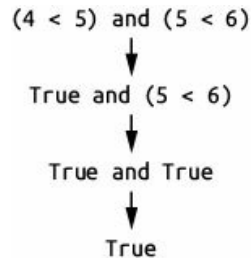



Figura 2.2 – O processo de avaliação de (4 < 5) and (5 < 6) *como* True.

Podemos também usar vários operadores booleanos em uma expressão, juntamente com os operadores de comparação.

```
>>> 2 + 2 == 4 and not 2 + 2 == 5 and 2 * 2 == 2 + 2
True
```

Os operadores booleanos têm uma ordem de atuação, assim como os operadores matemáticos. Depois que todos os operadores matemáticos e de comparação forem avaliados, o Python avaliará os operadores not em primeiro lugar, em seguida os operadores and e, por fim, os operadores or.

Elementos do controle de fluxo

As instruções de controle de fluxo geralmente começam com uma parte chamada *condição*, e todas as condições são seguidas de um bloco de código chamado *cláusula*. Antes de conhecer as instruções de controle de fluxo específicas do Python, discutirei o que são uma condição e um bloco.

Condições

As expressões booleanas que vimos até agora poderiam ser todas consideradas condições, que é o mesmo que expressões; uma *condição* é somente um nome mais específico no contexto das instruções de controle de fluxo. As condições sempre são avaliadas como um valor booleano, ou seja, True ou False. Uma instrução de controle de fluxo decide o que fazer conforme sua condição seja True ou False, e quase toda instrução de controle de fluxo

utiliza uma condição.

Blocos de código

As linhas de código Python podem ser agrupadas em *blocos*. Podemos dizer em que ponto um bloco começa e termina a partir da indentação das linhas de código. Há três regras para blocos.

1. Os blocos começam no local em que a indentação aumenta.
2. Os blocos podem conter outros blocos.
3. Os blocos terminam no local em que a indentação se reduz a zero ou na indentação do bloco que o contém.

Os blocos são mais simples de entender se observarmos alguns códigos indentados, portanto vamos identificar os blocos em parte de um pequeno programa de jogo mostrado a seguir:

```
  if name == 'Mary':
u     print('Hello Mary')
  if password == 'swordfish':
v     print('Access granted.')
  else:
w     print('Wrong password.')
```

O primeiro bloco de código u começa na linha print('Hello Mary') e contém todas as linhas depois dela. Nesse bloco, temos outro bloco v, que tem apenas uma única linha: print('Access Granted.'). O terceiro bloco w também tem somente uma linha: print('Wrong password.').

Execução do programa

No programa *hello.py* do capítulo anterior, o Python começava executando as instruções do início do programa até o final, uma instrução após a outra. A *execução do programa* (ou simplesmente *execução*) é um termo que se refere à instrução atual sendo executada. Se você imprimir o código-fonte no papel e colocar seu dedo em cada linha à medida que esta for executada, podemos pensar em seu dedo como a execução do programa.

Nem todos os programas, porém, executam simplesmente de cima para

baixo. Se você usar seu dedo para seguir um programa contendo instruções de controle de fluxo, é provável que você dê alguns saltos pelo código-fonte de acordo com as condições e, provavelmente, pule cláusulas inteiras.

Instrução de controle de fluxo

Vamos agora explorar a parte mais importante do controle de fluxo: as instruções propriamente ditas. As instruções representam os losangos que vimos no fluxograma da figura 2.1 e correspondem às decisões que seu programa tomará.

Instruções if

O tipo mais comum de instrução de controle de fluxo é a instrução if. A cláusula de uma instrução if (ou seja, o bloco após a instrução if) será executada se a condição da instrução for True. A cláusula será ignorada se a condição for False.

Falando explicitamente, uma instrução if pode ser lida como “se (if) esta condição for verdadeira, execute o código que está na cláusula”. Em Python, uma instrução if é constituída das seguintes partes:

- a palavra-chave if;
- uma condição (ou seja, uma expressão avaliada como True ou False);
- dois-pontos;
- começando na próxima linha, um bloco de código indentado (chamado de cláusula if).

Por exemplo, suponha que temos um código que verifique se o nome de uma pessoa é Alice. (Imagine que name tenha recebido um valor anteriormente.)

```
if name == 'Alice':
    print('Hi, Alice.')
```

Todas as instruções de controle de fluxo terminam com dois pontos e são seguidas por um novo bloco de código (a cláusula). A cláusula dessa instrução if é o bloco com print('Hi, Alice.'). A figura 2.3 mostra a aparência de um fluxograma desse código.

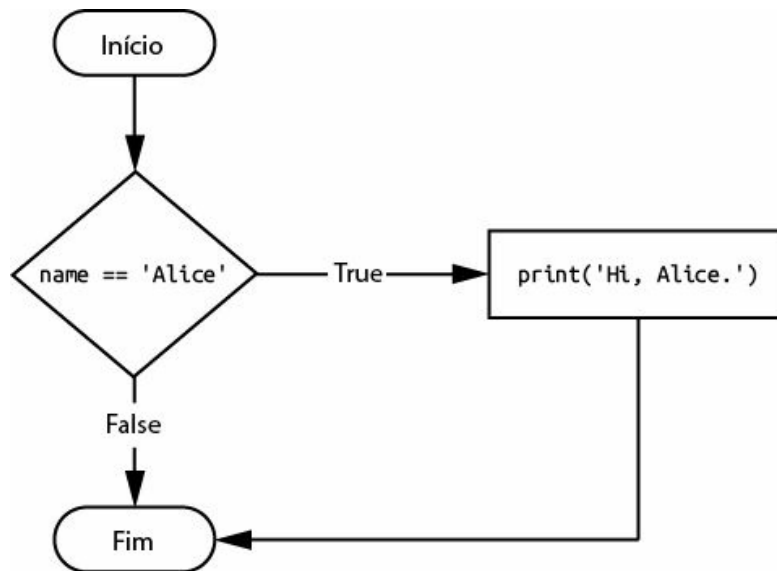



Figura 2.3 – O fluxograma de uma instrução if.

Instruções else

Uma cláusula if pode opcionalmente ser seguida de uma instrução else. A cláusula else será executada somente quando a condição da instrução if for False. Falando explicitamente, uma instrução else pode ser lida como "se esta condição for verdadeira, execute este código; senão (else) execute aquele código". Uma instrução else não tem uma condição e, no código, ela sempre será constituída das seguintes partes:

- a palavra-chave else;
- dois-pontos;
- começando na próxima linha, um bloco de código indentado (chamado de cláusula else).

Retornando ao exemplo com Alice, vamos dar uma olhada em um código que utiliza uma instrução else para oferecer uma saudação diferente caso o nome da pessoa não seja Alice.

```
if name == 'Alice':
    print('Hi, Alice.')
else:
    print('Hello, stranger.')
```

A figura 2.4 mostra a aparência de um fluxograma desse código.

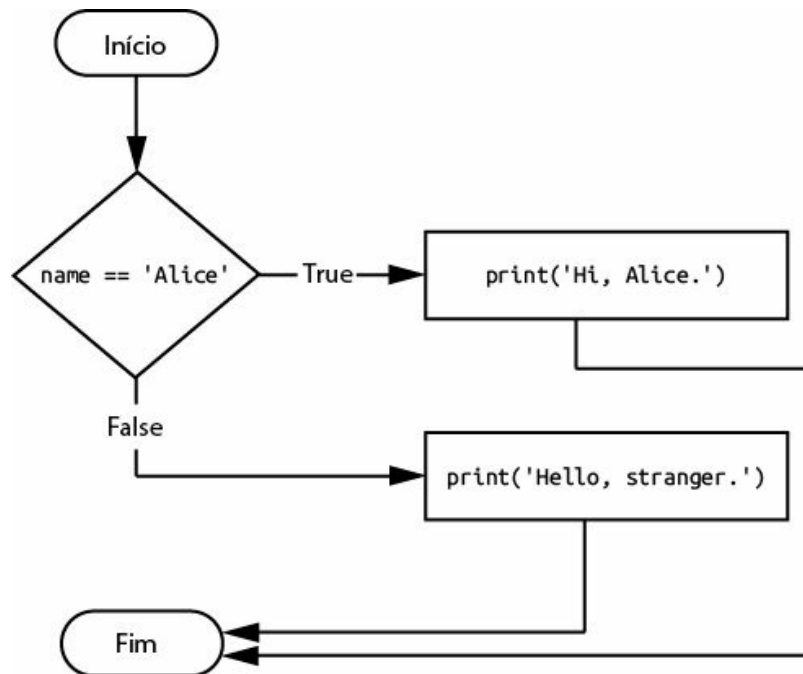



Figura 2.4 – O fluxograma de uma instrução else*.*

Instruções elif

Embora somente uma das cláusulas if ou else seja executada, você poderá ter um caso em que queira que uma entre *várias* cláusulas possíveis seja executada. A instrução elif é uma instrução "else if" que sempre vem após um if ou outra instrução elif. Ela provê outra condição que será verificada somente se todas as condições anteriores forem False. No código, uma instrução elif será sempre constituída das seguintes partes:

- a palavra-chave elif;
- uma condição (ou seja, uma expressão avaliada como True ou False);
- dois-pontos;
- começando na próxima linha, um bloco de código indentado (chamado de cláusula elif).

Vamos acrescentar um elif ao verificador de nomes e ver essa instrução em ação.

```
if name == 'Alice':
    print('Hi, Alice.')
elif age < 12:
    print('You are not Alice, kiddo.')
```

Dessa vez, verificamos a idade da pessoa e o programa informará algo diferente se essa pessoa tiver menos de 12 anos. Podemos ver o fluxograma desse programa na figura 2.5.

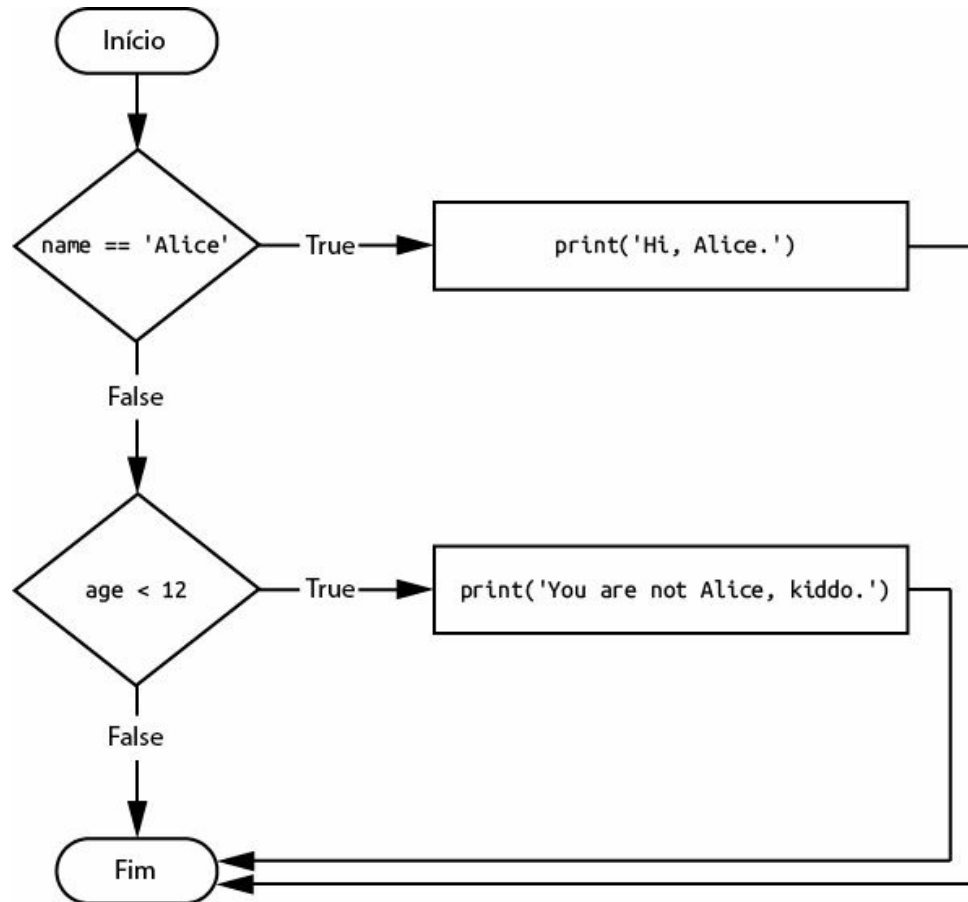



Figura 2.5 – O fluxograma de uma instrução elif.

A cláusula elif executará se age < 12 for igual a True e name == 'Alice' for False. No entanto, se ambas as condições forem False, as duas cláusulas serão ignoradas. *Não* há garantias de que pelo menos uma das cláusulas seja executada. Quando houver uma cadeia de instruções elif, somente uma ou nenhuma das cláusulas será executada. Se for determinado que a condição de uma das instruções é True, o restante das cláusulas elif será automaticamente ignorado. Por exemplo, abra uma nova janela no editor de arquivo e insira o código a seguir, salvando-o como *vampire.py*:

```
if name == 'Alice':
    print('Hi, Alice.')
elif age < 12:
```

```
    print('You are not Alice, kiddo.')
elif age > 2000:
    print('Unlike you, Alice is not an undead, immortal vampire.')
elif age > 100:
    print('You are not Alice, grannie.')
```

Nesse caso, adicionamos mais duas instruções elif para fazer o verificador de nomes saudar uma pessoa com diferentes respostas de acordo com o valor de age. A figura 2.6 mostra o fluxograma desse código.

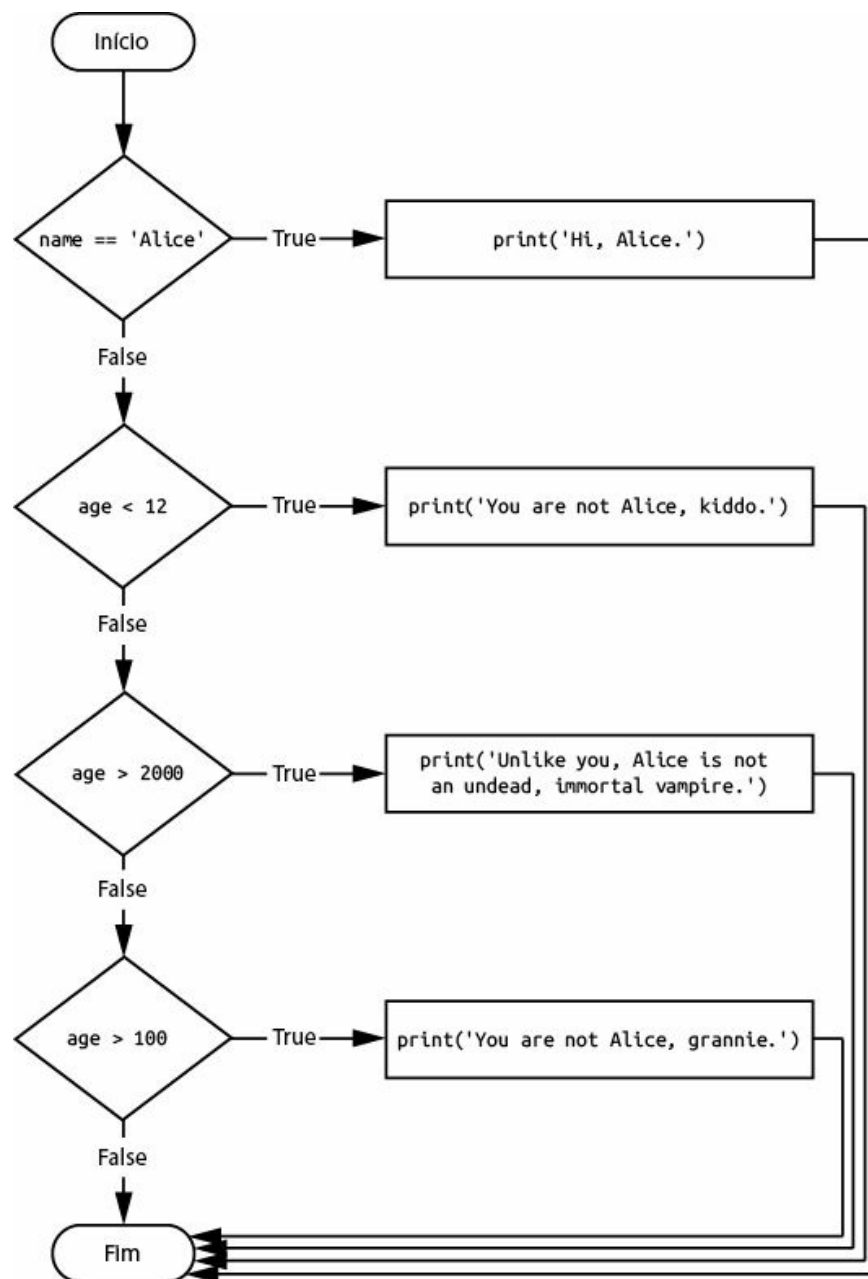



Figura 2.6 – O fluxograma para várias instruções elif *no programa vampire.py.*

Entretanto a ordem das instruções elif é importante. Vamos reorganizá-las de modo a introduzir um bug. Lembre-se de que o restante das cláusulas elif é automaticamente ignorado após uma condição True ser determinada, portanto, se você trocar algumas das cláusulas de *vampire.py*, um problema poderá ocorrer. Altere o código para que tenha a aparência a seguir e salve-o como *vampire2.py*:

```
  if name == 'Alice':
      print('Hi, Alice.')
  elif age < 12:
      print('You are not Alice, kiddo.')
u elif age > 100:
      print('You are not Alice, grannie.')
  elif age > 2000:
      print('Unlike you, Alice is not an undead, immortal vampire.')
```

Suponha que a variável age contenha o valor 3000 antes de esse código ser executado. Você poderá esperar que o código exiba a string 'Unlike you, Alice is not an undead, immortal vampire.' (diferente de você, Alice não é uma vampira morta-viva e imortal). Entretanto, como a condição age > 100 é True (afinal de contas, 3000 *é* maior que 100) u, a string 'You are not Alice, grannie.' (Você não é Alice, vovó.) será exibida e o restante das instruções elif será automaticamente ignorado. Lembre-se de que no máximo uma das cláusulas será executada e, para as instruções elif, a ordem importa!

A figura 2.7 mostra o fluxograma do código anterior. Observe como os losangos para age > 100 e age > 2000 foram trocados.

Opcionalmente, podemos ter uma instrução else após a última instrução elif. Nesse caso, *há* garantias de que pelo menos uma (e somente uma) das cláusulas seja executada. Se as condições em todas as instruções if e elif forem False, a cláusula else será executada. Por exemplo, vamos recriar o programa de Alice de modo a usar cláusulas if, elif e else.

```
if name == 'Alice':
    print('Hi, Alice.')
elif age < 12:
    print('You are not Alice, kiddo.')
```

```
else:
    print('You are neither Alice nor a little kid.')
```

A figura 2.8 mostra o fluxograma desse novo código, que salvaremos como *littleKid.py*.

Falando explicitamente, esse tipo de estrutura de controle de fluxo corresponde a "Se a primeira condição for verdadeira, faça isso; senão, se a segunda condição for verdadeira, faça aquilo. Do contrário, faça algo diferente.". Quando essas três instruções forem usadas em conjunto, lembre-se dessas regras sobre como ordená-las para evitar bugs como aquele da figura 2.7. Inicialmente, sempre deve haver exatamente uma instrução if. Qualquer instrução elif necessária deverá vir após a instrução if. Em segundo lugar, se quiser ter certeza de que pelo menos uma cláusula será executada, encerre a estrutura com uma instrução else.

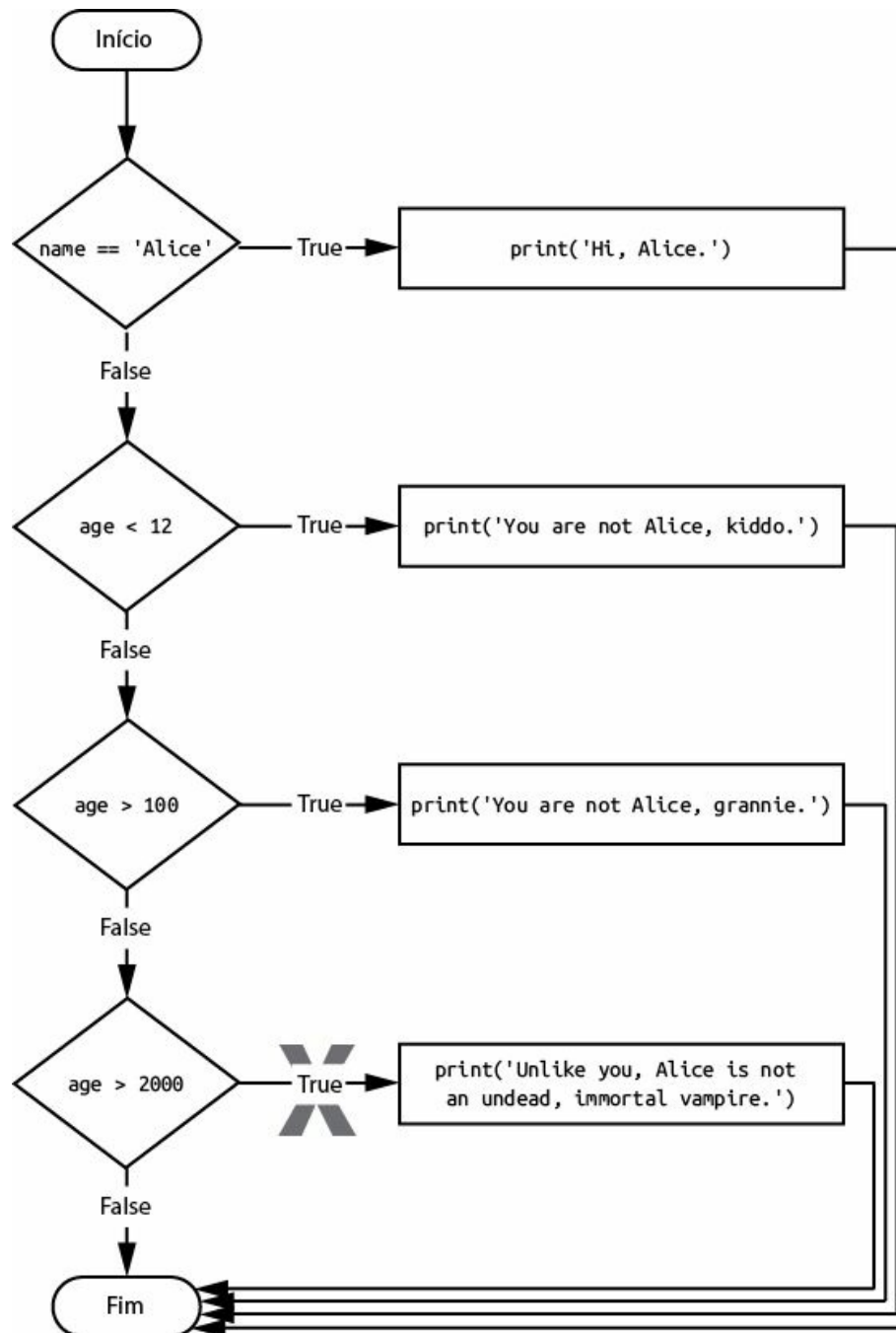



Figura 2.7 – O fluxograma do programa vampire2.py. O caminho com um X jamais ocorrerá do ponto de vista lógico, pois se age *for maior que* 2000*, ele já será maior que* 100*.*

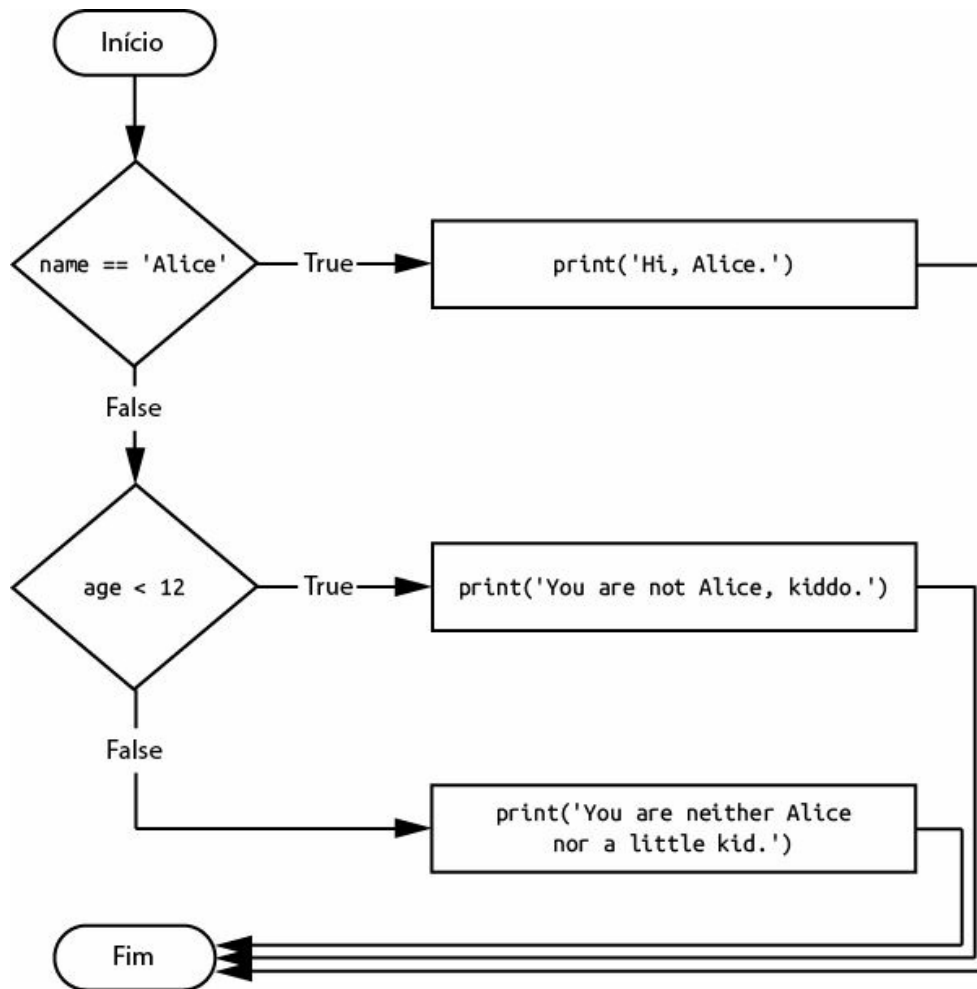



Figura 2.8 – Fluxograma do programa littleKid.py anterior.

Instruções de loop while

Podemos fazer um bloco de código ser executado repetidamente usando uma instrução while. O código em uma cláusula while será executado enquanto a condição da instrução while for True. No código, uma instrução while sempre será constituída das seguintes partes:

- a palavra-chave while;
- uma condição (ou seja, uma expressão avaliada como True ou False);
- dois-pontos;
- começando na próxima linha, um bloco de código indentado (chamado de cláusula while).

Podemos ver que uma instrução while é semelhante a uma instrução if. A

diferença está em seu comportamento. No final de uma cláusula if, a execução do programa continua após a instrução if. Porém, no final de uma cláusula while, a execução do programa retorna ao início da instrução while. A cláusula while geralmente é chamada de *loop while* ou simplesmente *loop*.

Vamos dar uma olhada em uma instrução if e em um loop while que usem a mesma condição e executem as mesmas ações de acordo com essa condição. Eis o código com uma instrução if:

```
spam = 0
if spam < 5:
    print('Hello, world.')
    spam = spam + 1
```

A seguir temos o código com uma instrução while:

```
spam = 0
while spam < 5:
    print('Hello, world.')
    spam = spam + 1
```

Essas instruções são semelhantes – tanto if quanto while verificam spam e, se o seu valor for menor que cinco, uma mensagem será exibida. No entanto, quando esses dois trechos de código são executados, algo muito diferente acontece em cada caso. Para a instrução if, a saída é simplesmente "Hello, world.". Porém, para a instrução while, é "Hello, world." repetido cinco vezes! Dê uma olhada nos fluxogramas para esses dois trechos de código nas figuras 2.9 e 2.10 e veja por que isso acontece.

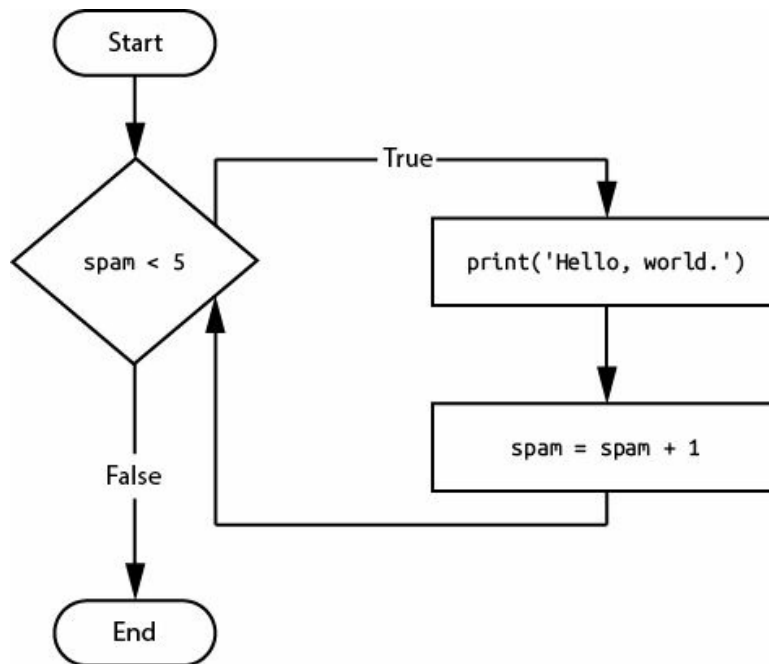



Figura 2.9 – O fluxograma do código com a instrução if.

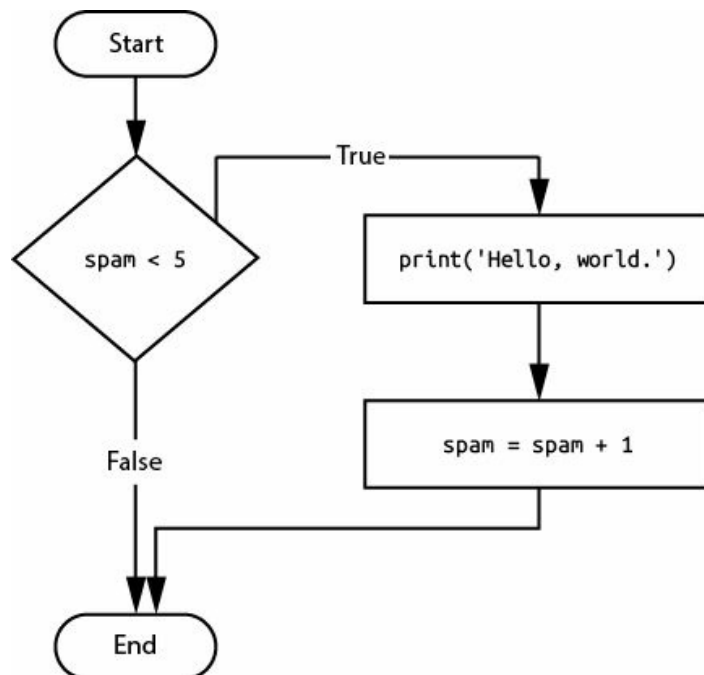



Figura 2.10 – O fluxograma do código com a instrução while.

O código com a instrução if verifica a condição e exibe Hello, world. somente uma vez se essa condição for verdadeira. O código com o loop while, por outro lado, exibirá essa string cinco vezes. Ele para após cinco exibições porque o inteiro em spam é incrementado de um no final de cada iteração do

loop, o que significa que o loop executará cinco vezes antes de spam < 5 ser False.

No loop while, a condição é sempre verificada no início de cada *iteração* (ou seja, sempre que o loop for executado). Se a condição for True, a cláusula será executada e, depois disso, a condição será verificada novamente. Na primeira vez que se verificar que a condição é False, a cláusula while será ignorada.

Um loop while irritante

A seguir, apresentamos um pequeno exemplo de programa que ficará pedindo para você digitar **your name** literalmente. Selecione File4New Window (Arquivo4Nova janela) para abrir uma nova janela no editor de arquivo, digite o código a seguir e salve o arquivo como *yourName.py*:

```
u name = ''
v while name != 'your name':
      print('Please type your name.')
w     name = input()
x print('Thank you!')
```

Inicialmente, o programa define a variável name u com uma string vazia. Isso serve para que a condição name != 'your name' seja avaliada como True e a execução do programa entre na cláusula do loop while v.

O código nessa cláusula pede ao usuário que digite seu nome, que é atribuído à variável name w. Como essa é a última linha do bloco, a execução retorna ao início do loop while e avalia a condição novamente. Se o valor em name for *diferente* da string 'your name', a condição será True e a execução entrará novamente na cláusula while.

Porém, quando o usuário digitar **your name**, a condição do loop while será 'your name' != 'your name', que será avaliada como False. A condição agora é False e, em vez de entrar novamente na cláusula do loop while, a execução do programa a ignorará e o restante do programa será executado x. A figura 2.11 mostra um fluxograma do programa *yourName.py*.

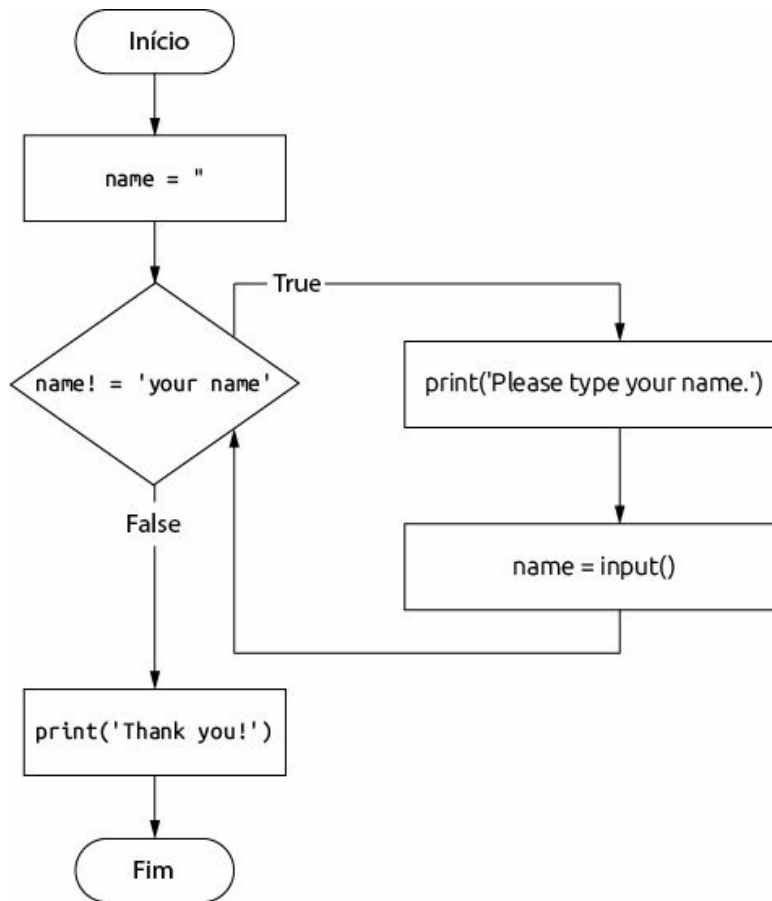



Figura 2.11 – Um fluxograma do programa yourName.py.

Agora vamos ver *yourName.py* em ação. Tecle F5 para executá-lo e digite algo diferente de **your name** algumas vezes, antes de fornecer ao programa o que ele quer.

```
Please type your name.
Al
Please type your name.
Albert
Please type your name.
%#@#%*(^&!!!
Please type your name.
your name
Thank you!
```

Se você não digitar **your name**, a condição do loop while jamais será False e o programa simplesmente continuará fazendo o seu pedido para sempre. Nesse caso, a chamada a input() permite que o usuário forneça a string correta para

fazer o programa continuar. Em outros programas, a condição talvez jamais mude e isso poderá ser um problema. Vamos dar uma olhada em como sair de um loop while.

Instruções break

Há um atalho para fazer a execução do programa sair previamente de uma cláusula de loop while. Se uma instrução break for alcançada, a execução sairá imediatamente da cláusula do loop while. No código, uma instrução break simplesmente contém a palavra-chave break.

Bem simples, não? A seguir, apresentamos um programa que faz o mesmo que o programa anterior, porém utiliza uma instrução break para sair do loop. Digite o código a seguir e salve o arquivo como *yourName2.py*:

```
u while True:
      print('Please type your name.')
v     name = input()
w     if name == 'your name':
x         break
y print('Thank you!')
```

A primeira linha u cria um *loop infinito*; é um loop while cuja condição é sempre True. (A expressão True, afinal de contas, é sempre avaliada com o valor True.) A execução do programa sempre entrará no loop e sairá somente quando uma instrução break for executada. (Um loop infinito que não termina *nunca* é um bug comum de programação.)

Como antes, esse programa pede que o usuário digite **your name** v. Agora, porém, enquanto a execução ainda está no loop while, uma instrução if é executada w para verificar se name é igual a your name. Se essa condição for True, a instrução break será executada x e a execução sairá do loop para exibir print('Thank you!') y. Caso contrário, a cláusula da instrução if com a instrução break será ignorada, o que colocará a execução no final do loop while. A essa altura, a execução do programa retorna ao início da instrução while u para verificar novamente a condição. Como essa condição é simplesmente o valor booleano True, a execução entra no loop para pedir ao usuário que digite **your name** novamente. Veja a figura 2.12, que contém o fluxograma desse programa.

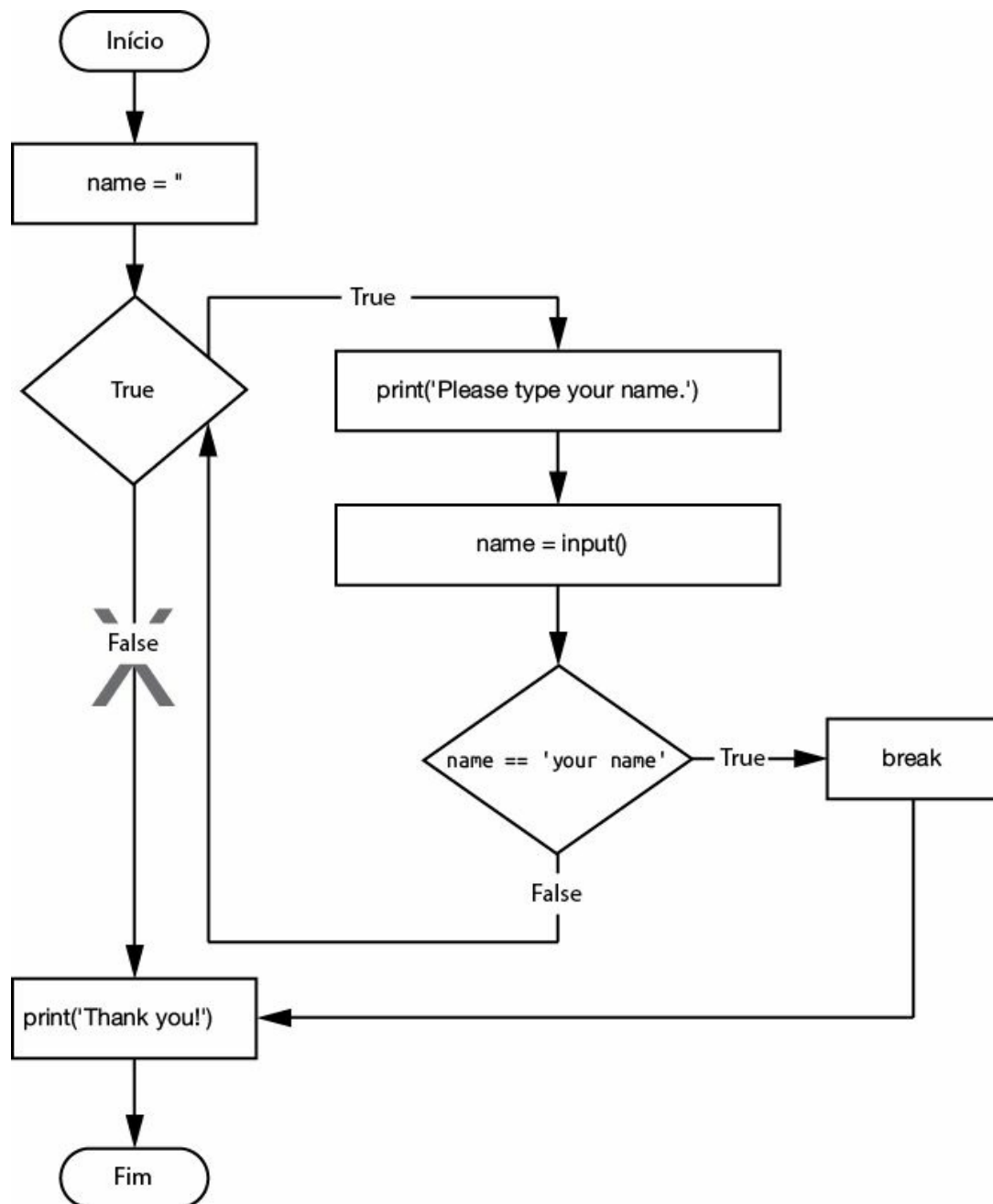



Figura 2.12 – O fluxograma do programa yourName2.py *com um loop infinito. Observe que o caminho com X jamais ocorrerá do ponto de vista lógico, pois a condição do loop sempre será* True.

Execute *yourName2.py* e digite o mesmo texto fornecido a *yourName.py*. O programa reescrito deverá responder da mesma maneira que o programa original.

Instruções continue

Assim como as instruções break, as instruções continue são usadas nos loops. Quando alcançar uma instrução continue, a execução do programa retornará imediatamente ao início do loop e a condição será avaliada novamente. (Isso é o que também acontece quando a execução alcança o final do loop.)

PRESO EM UM LOOP INFINITO?

Se você executar um programa com um bug que o faça ficar preso em um loop infinito, tecle CTRL-C. Isso enviará um erro KeyboardInterrupt ao seu programa e o fará ser interrompido imediatamente. Para testar isso, crie um loop infinito simples no editor de arquivo e salve-o como *infiniteloop.py*.

```
while True:
    print('Hello world!')
```

Ao executar esse programa, Hello world! será exibido na tela indefinidamente porque a condição da instrução while será sempre True. Na janela de shell interativo do IDLE, há somente duas maneiras de interromper esse programa: teclar CTRL-C ou selecionar Shell4Restart Shell (Shell4Reiniciar shell) no menu. CTRL-C será conveniente se você quiser encerrar seu programa imediatamente, mesmo que ele não esteja preso em um loop infinito.

Vamos utilizar continue para escrever um programa que peça um nome e uma senha. Digite o código a seguir em uma nova janela do editor de arquivo e salve o programa como *swordfish.py*.

```
  while True:
      print('Who are you?')
      name = input()
u     if name != 'Joe':
v         continue
      print('Hello, Joe. What is the password? (It is a fish.)')
w     password = input()
      if password == 'swordfish':
x         break
y print('Access granted.')
```

Se o usuário fornecer qualquer nome que não seja Joe u, a instrução continue v fará a execução do programa retornar ao início do loop. Quando a condição for reavaliada, a execução sempre entrará no loop, pois essa condição

corresponde simplesmente ao valor True. Se o usuário conseguir passar pela instrução if, uma senha será solicitada w. Se a senha fornecida for swordfish, a instrução break x será executada e a execução sairá do loop while para exibir Access granted y. Caso contrário, a execução continuará até o final do loop while, quando retornará ao início do loop. Veja a figura 2.13, que contém o fluxograma desse programa.

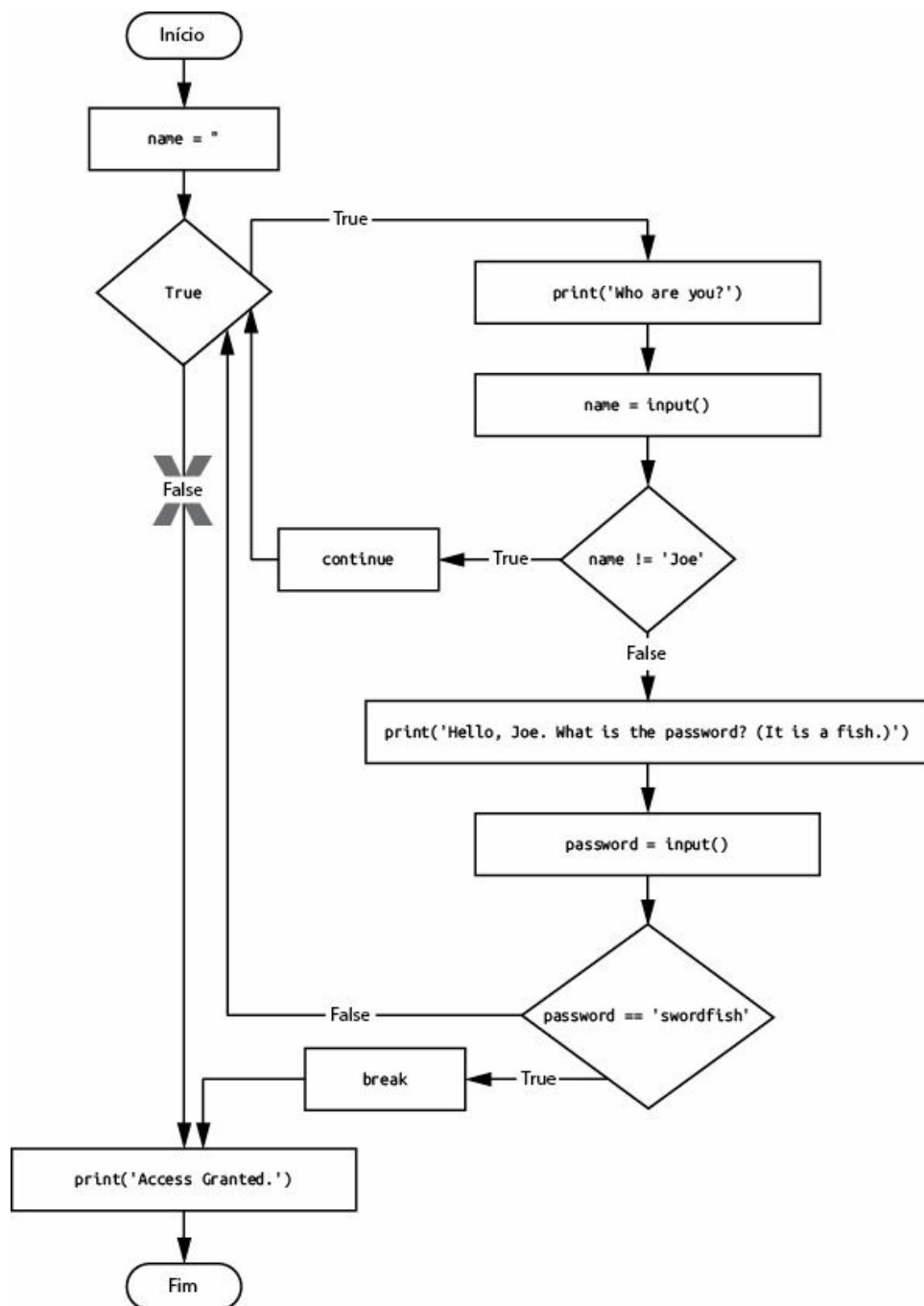



Figura 2.13 – Um fluxograma de swordfish.py*. O caminho com X jamais ocorrerá do ponto de vista lógico, pois a condição do loop sempre será* True*.*

VALORES "TRUTHY" E "FALSEY"

Há alguns valores de outros tipos de dados para os quais as condições os considerarão equivalentes a True e False. Quando usados em condições, 0, 0.0 e '' (a string vazia) são considerados False, enquanto todos os demais valores são considerados True. Por exemplo, observe o programa a seguir:

```
name = ''
while not name:u
    print('Enter your name:')
    name = input()
print('How many guests will you have?')
numOfGuests = int(input())
if numOfGuests:v
    print('Be sure to have enough room for all your guests.')w
print('Done')
```

Se o usuário fornecer uma string vazia para name, a condição da instrução while será True u e o programa continuará pedindo um nome. Se o valor de numOfGuests for diferente de 0 v, a condição será considerada True e o programa exibirá um lembrete ao usuário w.

Você poderia ter digitado not name != '' no lugar de not name e numOfGuests != 0 no lugar de numOfGuests, porém usar os valores truthy e falsey podem deixar seu código mais legível.

Execute esse programa e forneça-lhe alguns dados de entrada. Até você afirmar que é Joe, ele não deverá pedir uma senha e, depois que a senha correta for fornecida, o programa deverá sair.

Who are you?
I'm fine, thanks. Who are you?
Who are you?
Joe
Hello, Joe. What is the password? (It is a fish.)
Mary
Who are you?
Joe

```
Hello, Joe. What is the password? (It is a fish.)
swordfish
Access granted.
```

Loops for e a função range()

O loop while continuará executando enquanto sua condição for True (que é o motivo para se chamar while, que quer dizer "enquanto"), mas o que aconteceria se você quisesse executar um bloco de código somente um determinado número de vezes? Podemos fazer isso usando uma instrução de loop for e a função range().

No código, uma instrução for é semelhante a for i in range(5): e sempre inclui as partes a seguir:

- a palavra-chave for;
- um nome de variável;
- a palavra-chave in;
- uma chamada ao método range() com até três inteiros passados a ele;
- dois-pontos;
- começando na próxima linha, um bloco de código indentado (chamado de cláusula for).

Vamos criar um novo programa chamado *fiveTimes.py* para ajudar você a ver um loop for em ação.

```
print('My name is')
for i in range(5):
    print('Jimmy Five Times (' + str(i) + ')')
```

O código da cláusula do loop for é executado cinco vezes. Na primeira vez que é executado, a variável i é definida com 0. A chamada a print() na cláusula exibirá Jimmy Five Times (0). Depois que o Python finalizar uma iteração passando por todo o código da cláusula do loop for, a execução retornará ao início do loop e a instrução for incrementará i de um. É por isso que range(5) resulta em cinco iterações pela cláusula, com i sendo definido com 0, em seguida com 1, 2, 3 e, por fim, 4. A variável i assumirá os valores até o inteiro passado para range(), porém sem incluí-lo. A figura 2.14 mostra um fluxograma do programa *fiveTimes.py*.

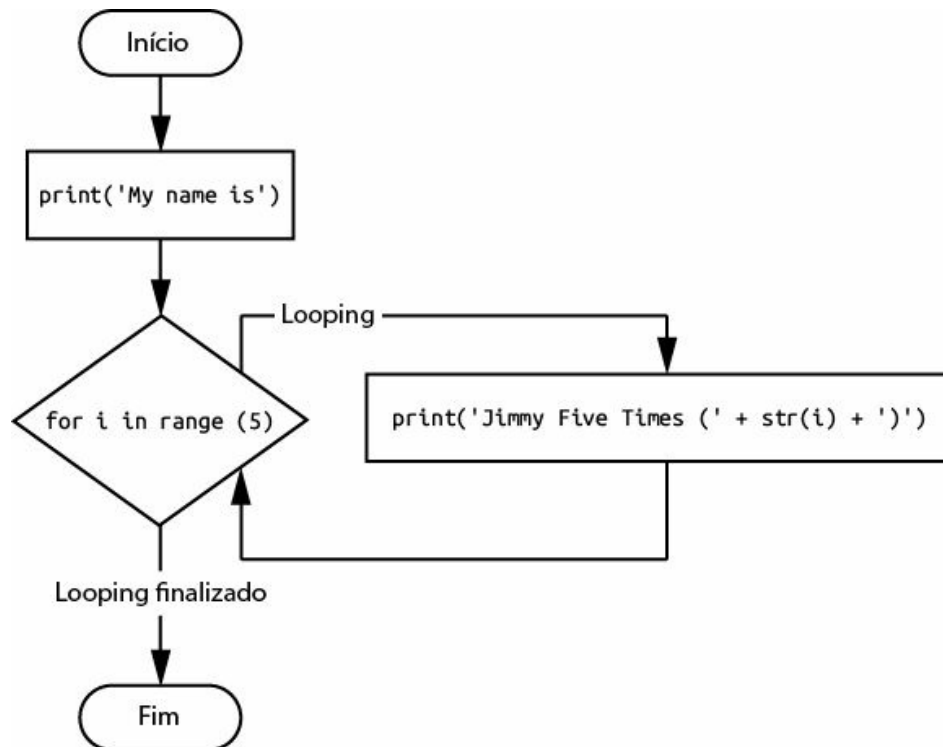



Figura 2.14 – O fluxograma de fiveTimes.py.

Ao ser executado, esse programa deverá exibir Jimmy Five Times seguido do valor de i cinco vezes, antes de sair do loop for.

```
My name is
Jimmy Five Times (0)
Jimmy Five Times (1)
Jimmy Five Times (2)
Jimmy Five Times (3)
Jimmy Five Times (4)
```

NOTA As instruções break e continue também podem ser usadas em loops for. A instrução continue continuará com o próximo valor do contador do loop for, como se a execução do programa tivesse alcançado o final do loop e retornado ao início. De fato, as instruções continue e break podem ser usadas somente em loops while e for. Se você tentar usar essas instruções em outros lugares, o Python exibirá uma mensagem de erro.

Como outro exemplo de loop for, considere esta história sobre o matemático Karl Friedrich Gauss. Quando Gauss ainda era criança, um professor queria dar uma tarefa que deixasse a classe ocupada. O professor disse aos alunos para que somassem todos os números de 0 a 100. O jovem Gauss concebeu um truque inteligente para descobrir a resposta em alguns segundos, porém

você pode criar um programa Python com um loop for para fazer esse cálculo.

```
u total = 0
v for num in range(101):
w     total = total + num
x print(total)
```

O resultado deve ser 5.050. Quando o programa inicia, a variável total é definida com 0 u. O loop for v então executa total = total + num w 100 vezes. Quando o loop tiver terminado de executar todas as suas 100 iterações, todos os inteiros de 0 a 100 terão sido somados a total. Nesse momento, total será exibido na tela x. Mesmo nos computadores mais lentos, esse programa exigirá menos de um segundo para ser concluído.

(O jovem Gauss descobriu que havia 50 pares de números que somavam 100: 1 + 99, 2 + 98, 3 + 97 e assim por diante, até 49 + 51. Como 50 × 100 é igual a 5.000, ao somar aquele 50 intermediário, a soma de todos os números de 0 a 100 será 5.050. Garoto esperto!)

Um equivalente ao loop while

Na verdade, podemos usar um loop while para fazer o mesmo que um loop for; os loops for são simplesmente mais concisos. Vamos reescrever *fiveTimes.py* para que utilize um loop while equivalente a um loop for.

```
print('My name is')
i = 0
while i < 5:
    print('Jimmy Five Times (' + str(i) + ')')
    i = i + 1
```

Ao executar esse programa, a saída deverá ser semelhante àquela do programa *fiveTimes.py* que utiliza um loop for.

Argumentos de início, fim e de incremento de range()

Algumas funções podem ser chamadas com vários argumentos separados por uma vírgula, e range() é uma delas. Isso permite alterar o inteiro passado para range() de modo a seguir qualquer sequência de inteiros, incluindo começar com um número diferente de zero.

```
for i in range(12, 16):
    print(i)
```

O primeiro argumento será o ponto em que a variável do loop for deve iniciar; o segundo argumento será o número com o qual o loop deve terminar, sem incluí-lo.

```
12
13
14
15
```

A função range() também pode ser chamada com três argumentos. Os dois primeiros argumentos serão os valores de início e fim, e o terceiro será o *argumento de incremento*. O incremento é a quantidade somada à variável após cada iteração.

```
for i in range(0, 10, 2):
    print(i)
```

Chamar range(0, 10, 2) fará uma contagem de zero a oito em intervalos de dois.

```
0
2
4
6
8
```

A função range() é flexível quanto à sequência de números gerados em loops for. Por exemplo, você pode até mesmo usar um número negativo para o argumento de incremento para fazer o contador do loop for diminuir em vez de aumentar.

```
for i in range(5, -1, -1):
    print(i)
```

A execução de um loop for para exibir i com range(5, -1, -1) deverá exibir de cinco a zero.

```
5
4
```

```
3
2
1
0
```

Importando módulos

Todos os programas Python podem chamar um conjunto básico de funções denominado *funções internas* (built-in), incluindo as funções print(), input() e len(), que vimos anteriormente. O Python também vem com um conjunto de módulos chamado *biblioteca-padrão* (standard library). Cada módulo corresponde a um programa Python que contém um grupo relacionado de funções que pode ser incluído em seus programas. Por exemplo, o módulo math contém funções relacionadas à matemática, o módulo random contém funções relacionadas a números aleatórios, e assim por diante.

Antes de poder usar as funções de um módulo, você deve importá-lo usando uma instrução import. No código, uma instrução import é constituída das seguintes partes:

- a palavra-chave import;
- o nome do módulo;
- opcionalmente, mais nomes de módulos, desde que estejam separados por vírgula.

Após ter importado um módulo, você poderá usar todas as funções interessantes desse módulo. Vamos testar isso com o módulo random, que nos dará acesso à função random.randint().

Digite o código a seguir em seu editor de arquivo e salve-o como *printRandom.py*:

```
import random
for i in range(5):
   print(random.randint(1, 10))
```

Ao executar esse programa, a saída terá uma aparência semelhante a:

```
4
1
8
```

```
4
1
```

A chamada à função random.randint() é avaliada como um valor inteiro aleatório entre os dois inteiros passados a ela. Como randint() está no módulo random, você deve digitar **random.** na frente do nome da função para informar o Python que essa função deve ser procurada no módulo random.

Eis um exemplo de uma instrução import que importa quatro módulos diferentes:

```
import random, sys, os, math
```

Agora podemos utilizar qualquer uma das funções contidas nesses quatro módulos. Aprenderemos mais sobre esse assunto posteriormente neste livro.

Instruções from import

Uma forma alternativa para a instrução import é composta da palavra-chave from seguida do nome do módulo, a palavra-chave import e um asterisco; por exemplo, from random import *.

Com essa forma da instrução import, as chamadas às funções em random não precisarão do prefixo random. Entretanto usar o nome completo deixa o código mais legível, portanto é melhor usar a forma normal da instrução import.

Encerrando um programa previamente com sys.exit()

O último conceito de controle de fluxo a ser discutido relaciona-se ao modo de encerrar o programa. Isso sempre acontece quando a execução do programa alcança o fim das instruções. No entanto podemos fazer o programa ser encerrado, isto é, sair do programa, chamando a função sys.exit(). Como essa função está no módulo sys, devemos importar sys antes de o seu programa poder usá-lo.

Abra uma nova janela no editor de arquivo e insira o código a seguir, salvando-o como *exitExample.py*:

```
import sys
```

```
while True:
    print('Type exit to exit.')
    response = input()
    if response == 'exit':
        sys.exit()
    print('You typed ' + response + '.')
```

Execute esse programa no IDLE. O programa contém um loop infinito sem uma instrução break. A única maneira de esse programa terminar será o usuário digitar exit, fazendo sys.exit() ser chamado. Quando response for igual a exit, o programa será encerrado. Como a variável response é definida pela função input(), o usuário deverá fornecer exit para interromper o programa.

Resumo

Ao usar expressões que são avaliadas como True ou False (também chamadas de *condições*), podemos criar programas que tomem decisões em relação ao código a ser executado e ignorado. Também podemos executar códigos repetidamente em um loop enquanto determinada condição for avaliada como True. As instruções break e continue serão úteis caso você precise sair de um loop ou voltar para o início.

Essas instruções de controle de fluxo permitirão escrever programas muito mais inteligentes. Há outro tipo de controle de fluxo que pode ser implementado quando criamos nossas próprias funções; esse será o assunto do próximo capítulo.

Exercícios práticos

1. Quais são os dois valores do tipo de dado booleano? Como eles são escritos?
2. Quais são os três operadores booleanos?
3. Escreva as tabelas verdade de cada operador booleano (ou seja, todas as combinações possíveis de valores booleanos para o operador e como elas são avaliadas).
4. Para que valores as expressões a seguir são avaliadas?

```
(5 > 4) and (3 == 5)
not (5 > 4)
(5 > 4) or (3 == 5)
not ((5 > 4) or (3 == 5))
(True and True) and (True == False)
(not False) or (not True)
```

5. Quais são os seis operadores de comparação?
6. Qual é a diferença entre o operador “igual a” e o operador de atribuição?
7. Explique o que é uma condição e quando você usaria uma.
8. Identifique os três blocos no código a seguir:

```
spam = 0
if spam == 10:
    print('eggs')
    if spam > 5:
        print('bacon')
    else:
        print('ham')
    print('spam')
print('spam')
```

9. Escreva um código que exiba Hello se 1 estiver armazenado em spam, Howdy se 2 estiver armazenado em spam e Greetings! se outro valor estiver armazenado em spam.
10. Que tecla você deve pressionar se o seu programa estiver preso em um loop infinito?
11. Qual é a diferença entre break e continue?
12. Qual é a diferença entre range(10), range(0, 10) e range(0, 10, 1) em um loop for?
13. Crie um pequeno programa que mostre os números de 1 a 10 usando um loop for. Em seguida, crie um programa equivalente que mostre os números de 1 a 10 usando um loop while.

14. Se você tivesse uma função chamada bacon() em um módulo chamado spam, como você a chamaria após ter importado spam?

Pontos extras: Dê uma olhada nas funções round() e abs() na Internet e descubra o que elas fazem. Faça experimentos com elas no shell interativo.

CAPÍTULO 3

FUNÇÕES

Você já tem familiaridade com as funções print(), input() e len() dos capítulos anteriores. O Python disponibiliza diversas funções internas (built-in) como essas, porém você pode criar suas próprias funções. Uma *função* é como um miniprograma dentro de um programa.

Para entender melhor o funcionamento das funções, vamos criar uma. Digite o programa a seguir no editor de arquivo e salve-o como *helloFunc.py*:

```
u def hello():
v     print('Howdy!')
      print('Howdy!!!')
      print('Hello there.')

w hello()
  hello()
  hello()
```

A primeira linha contém uma instrução def u que define uma função chamada hello(). O código no bloco após a instrução def v é o corpo da função. Esse código é executado quando a função é chamada, e não quando ela é inicialmente definida.

As linhas contendo hello() após a função w são chamadas à função. No código, uma chamada de função é constituída simplesmente do nome da função seguido de parênteses, possivelmente com alguns argumentos entre os parênteses. Quando alcança essas chamadas, a execução do programa segue para a linha inicial da função e começa a executar o código a partir daí. Ao alcançar o final da função, a execução retorna à linha em que a função foi chamada e continua percorrendo o código como anteriormente.

Pelo fato de esse programa chamar hello() três vezes, o código na função hello() será executado três vezes. Ao executar esse programa, a saída será semelhante a:

```
Howdy!
```

```
Howdy!!!
Hello there.
Howdy!
Howdy!!!
Hello there.
Howdy!
Howdy!!!
Hello there.
```

Um dos principais propósitos das funções consiste em agrupar códigos que serão executados diversas vezes. Sem uma função definida, seria necessário copiar e colar esse código cada vez que fosse usado e o programa teria o seguinte aspecto:

```
print('Howdy!')
print('Howdy!!!')
print('Hello there.')
print('Howdy!')
print('Howdy!!!')
print('Hello there.')
print('Howdy!')
print('Howdy!!!')
print('Hello there.')
```

Em geral, sempre evite duplicar códigos, pois, se algum dia você decidir atualizá-lo – por exemplo, se encontrar um bug que deva ser corrigido –, será preciso lembrar-se de alterar o código em todos os locais em que você o copiou.

À medida que adquirir mais experiência de programação, com frequência, você se verá removendo códigos duplicados, ou seja, que tenham sido copiados e colados (*deduplication*). Esse processo torna seus programas mais compactos, legíveis e fáceis de atualizar.

Instruções def com parâmetros

Ao chamar a função print() ou len(), passamos valores, chamados de *argumentos* nesse contexto, ao digitá-los entre os parênteses. Você também pode definir suas próprias funções que aceitem argumentos. Digite o exemplo a seguir no editor de arquivo e salve-o como *helloFunc2.py*:

```
u def hello(name):
v     print('Hello ' + name)

w hello('Alice')
  hello('Bob')
```

Ao executar esse programa, a saída terá o seguinte aspecto:

```
Hello Alice
Hello Bob
```

A definição da função hello() nesse programa contém um parâmetro chamado name u. Um *parâmetro* é uma variável em que um argumento é armazenado quando uma função é chamada. Na primeira vez que a função hello() é chamada, isso é feito com o argumento 'Alice' w. A execução do programa entra na função e a variável name é automaticamente definida com 'Alice', que é exibido pela instrução print() v.

Um aspecto em especial a ser observado sobre os parâmetros é que o valor armazenado em um parâmetro é esquecido quando a função retorna. Por exemplo, se adicionarmos print(name) depois de hello('Bob') no programa anterior, um NameError será gerado pelo programa, pois não há nenhuma variável chamada name. Essa variável foi destruída após o retorno da chamada à função hello('Bob'), portanto print(name) faria referência a uma variável name inexistente.

Isso é semelhante ao modo como as variáveis de um programa são esquecidas quando o programa termina. Falarei mais sobre o motivo pelo qual isso acontece mais adiante neste capítulo, quando discutirei o que é o escopo local de uma função.

Valores de retorno e instruções return

Quando chamamos a função len() e lhe passamos um argumento como 'Hello', a chamada à função será avaliada como o valor inteiro 5, que é o tamanho da string passada. Em geral, o valor com o qual uma chamada de função é avaliada é chamado de *valor de retorno* da função.

Ao criar uma função usando a instrução def, podemos especificar qual deverá ser o valor de retorno com uma instrução return. Uma instrução return é

constituída das seguintes partes:

• a palavra-chave return;

• o valor ou a expressão que a função deve retornar.

Quando uma expressão é usada com uma instrução return, o valor de retorno é aquele com o qual essa expressão é avaliada. Por exemplo, o programa a seguir define uma função que retorna uma string diferente de acordo com o número passado a ela como argumento. Digite o código seguinte no editor de arquivo e salve-o como *magic8Ball.py*:

```
u import random

v def getAnswer(answerNumber):
w     if answerNumber == 1:
          return 'It is certain'
      elif answerNumber == 2:
          return 'It is decidedly so'
      elif answerNumber == 3:
          return 'Yes'
      elif answerNumber == 4:
          return 'Reply hazy try again'
      elif answerNumber == 5:
          return 'Ask again later'
      elif answerNumber == 6:
          return 'Concentrate and ask again'
      elif answerNumber == 7:
          return 'My reply is no'
      elif answerNumber == 8:
          return 'Outlook not so good'
      elif answerNumber == 9:
          return 'Very doubtful'
x r = random.randint(1, 9)
y fortune = getAnswer(r)
z print(fortune)
```

Quando esse programa começa, o Python inicialmente importa o módulo random u. Em seguida, a função getAnswer() é definida v. Como a função está sendo definida (e não chamada), a execução ignorará esse código. A seguir, a função random.randint() é chamada com dois argumentos, 1 e 9 x. Ela é avaliada

como um inteiro aleatório entre 1 e 9 (incluindo os próprios 1 e 9) e esse valor será armazenado em uma variável chamada r.

A função getAnswer() é chamada com r como argumento y. A execução do programa segue para o início da função getAnswer() w e o valor r é armazenado em um parâmetro chamado answerNumber. Então, de acordo com esse valor em answerNumber, a função retorna um de vários possíveis valores de string. A execução do programa retorna para a linha que chamou getAnswer() originalmente, no final do programa y. A string retornada é atribuída a uma variável chamada fortune, que, por sua vez, é passada para uma chamada a print() z e é exibida na tela.

Observe que, como podemos passar valores de retorno como argumento para outra chamada de função, as três linhas a seguir

```
r = random.randint(1, 9)
fortune = getAnswer(r)
print(fortune)
```

podem ser compactadas para esta única linha equivalente:

```
print(getAnswer(random.randint(1, 9)))
```

Lembre-se de que as expressões são compostas de valores e de operadores. Uma chamada de função pode ser usada em uma expressão, pois ela será avaliada como o seu valor de retorno.

Valor None

Em Python, há um valor chamado None, que representa a ausência de um valor. None é o único valor do tipo de dado NoneType. (Outras linguagens de programação podem chamar esse valor de null, nil ou undefined.) Assim como os valores booleanos True e False, None deve ser digitado com uma letra *N* maiúscula.

Esse "valor sem valor" pode ser útil quando houver necessidade de armazenar algo que não deva ser confundido com um valor real em uma variável. Um local em que None é usado é como valor de retorno de print(). A função print() exibe um texto na tela, porém não precisa retornar nada, como len() ou input() o fazem. No entanto, como todas as chamadas de função

devem ser avaliadas com um valor de retorno, print() retorna None. Para ver isso em ação, digite o seguinte no shell interativo:

```
>>> spam = print('Hello!')
Hello!
>>> None == spam
True
```

Internamente, o Python acrescenta return None no final de qualquer definição de função que não tenha a instrução return. Isso é semelhante ao modo como um loop while ou for termina implicitamente com uma instrução continue. Além do mais, se uma instrução return sem valor for utilizada (ou seja, se houver somente a palavra-chave return sozinha), None será retornado.

Argumentos nomeados e print()

A maioria dos argumentos é identificada pela sua posição na chamada à função. Por exemplo, random.randint(1, 10) é diferente de random.randint(10, 1). A chamada de função random.randint(1, 10) retornará um inteiro aleatório entre 1 e 10 porque o primeiro argumento corresponde à extremidade inicial do intervalo e o segundo argumento é a extremidade final (ao passo que random.randint(10, 1) causará um erro).

Entretanto *argumentos nomeados* (keyword arguments) são identificados pela palavra-chave inserida antes deles na chamada à função. Os argumentos nomeados geralmente são usados para parâmetros opcionais. Por exemplo, a função print() contém os parâmetros opcionais end e sep para especificar o que deve ser exibido no final de seus argumentos e entre eles (separando-os), respectivamente.

Se o programa a seguir for executado

```
print('Hello')
print('World')
```

a saída terá o seguinte aspecto:

```
Hello
World
```

As duas strings aparecem em linhas separadas porque a função print()

adiciona automaticamente um caractere de quebra de linha no final da string recebida. No entanto podemos definir o argumento nomeado end para alterar esse valor para uma string diferente. Por exemplo, se o programa for

```
print('Hello', end='')
print('World')
```

a saída será semelhante a:

```
HelloWorld
```

A saída é exibida em uma única linha porque não há mais uma quebra de linha exibida após 'Hello'. Em vez disso, uma string vazia é exibida. Isso será conveniente se for necessário desabilitar a quebra de linha adicionada ao final de toda chamada à função print().

De modo semelhante, ao passar diversos valores de string a print(), a função as separará automaticamente com um único espaço. Digite o seguinte no shell interativo:

```
>>> print('cats', 'dogs', 'mice')
cats dogs mice
```

Porém você pode substituir a string default de separação passando o argumento nomeado sep. Digite o seguinte no shell interativo:

```
>>> print('cats', 'dogs', 'mice', sep=',')
cats,dogs,mice
```

Também podemos acrescentar argumentos nomeados às funções que criarmos; porém, antes disso, você deverá conhecer os tipos de dados lista e dicionário nos próximos dois capítulos. Por enquanto, basta saber que algumas funções têm argumentos nomeados opcionais que podem ser especificados quando a função é chamada.

Escopo local e global

Dizemos que as variáveis e os parâmetros atribuídos em uma função chamada existem no *escopo local* dessa função. Dizemos que as variáveis que recebem valor fora de todas as funções existem no *escopo global*. Uma variável que exista em um escopo local é chamada de *variável local*, enquanto uma

variável que exista no escopo global é chamada de *variável global*. Uma variável deve ser de um ou de outro tipo; ela não pode ser, ao mesmo tempo, local e global

Pense em um *escopo* como um contêiner para variáveis. Quando um escopo é destruído, todos os valores armazenados nas variáveis do escopo são esquecidos. Há somente um escopo global, e ele é criado quando seu programa inicia. Quando o programa termina, o escopo global é destruído e todas as suas variáveis são esquecidas. Se não fosse assim, na próxima vez que você executasse seu programa, as variáveis lembrariam os valores da última execução.

Um escopo local é criado sempre que uma função é chamada. Qualquer variável que receber um valor nessa função existirá no escopo local. Quando a função retornar, o escopo local será destruído e essas variáveis serão esquecidas. Na próxima vez que essa função for chamada, as variáveis locais não lembrarão os valores armazenados usados na última vez que a função foi chamada.

Os escopos são importantes por diversos motivos:

- O código no escopo global não pode usar nenhuma variável local.
- No entanto, um escopo local pode acessar variáveis globais.
- O código no escopo local de uma função não pode usar variáveis de nenhum outro escopo local.
- Podemos usar o mesmo nome para diferentes variáveis se elas estiverem em escopos distintos. Isso quer dizer que pode haver uma variável local chamada `spam` e uma variável global também chamada `spam`.

O motivo pelo qual o Python tem escopos diferentes em vez de transformar tudo em variáveis globais é que, quando as variáveis são modificadas pelo código de uma chamada em particular de uma função, essa função interagirá com o restante do programa somente por meio de seus parâmetros e do valor de retorno. Isso restringe a lista de linhas de código que possam estar provocando um bug. Se o seu programa não contivesse nada além de variáveis globais e tivesse um bug porque uma variável estava sendo definida com um valor inadequado, seria difícil identificar o local em que esse valor inadequado estava sendo definido. Essa variável poderia ter sido definida em

qualquer ponto do programa – e seu programa poderia ter centenas ou milhares de linhas! No entanto, se o bug ocorrer devido a uma variável local com um valor inadequado, você saberá que somente o código dessa função poderia tê-la definido incorretamente.

Embora usar variáveis globais em programas pequenos não seja um problema, contar com variáveis globais à medida que seus programas ficarem cada vez mais extensos é um péssimo hábito.

Variáveis locais não podem ser usadas no escopo global

Considere o programa a seguir, que causará um erro ao ser executado:

```
def spam():
    eggs = 31337
spam()
print(eggs)
```

Ao executar esse programa, a saída será semelhante a:

```
Traceback (most recent call last):
  File "C:/test3784.py", line 4, in <module>
    print(eggs)
NameError: name 'eggs' is not defined
```

O erro ocorre porque a variável eggs existe somente no escopo local criado quando spam() é chamado. Depois que a execução do programa retorna de spam, esse escopo local é destruído e não haverá mais uma variável chamada eggs. Sendo assim, quando seu programa tenta executar print(eggs), o Python apresentará um erro informando que eggs não está definido. Isso faz sentido se você pensar no assunto; quando a execução do programa está no escopo global, não há nenhum escopo local, portanto não poderá haver nenhuma variável local. É por isso que somente variáveis globais podem ser usadas no escopo global.

Escopos locais não podem usar variáveis de outros escopos locais

Um novo escopo local é criado sempre que uma função é chamada, inclusive quando uma função é chamada a partir de outra função. Considere o

programa a seguir:

```
   def spam():
u      eggs = 99
v      bacon()
w      print(eggs)

   def bacon():
       ham = 101
x      eggs = 0

y spam()
```

Quando o programa inicia, a função spam() é chamada y e um escopo local é criado. A variável local eggs u é definida com 99. Então a função bacon() é chamada v e um segundo escopo local é criado. Vários escopos locais podem existir ao mesmo tempo. Nesse novo escopo local, a variável local ham é definida com 101 e uma variável local eggs – que é diferente daquela do escopo local de spam() – também é criada x e definida com 0.

Quando bacon() retorna, o escopo local dessa chamada é destruído. A execução do programa continua na função spam() para exibir o valor de eggs w e, como o escopo local para a chamada a spam() continua existindo nesse ponto, a variável eggs é definida com 99. É esse valor que o programa exibe.

Podemos concluir que as variáveis locais de uma função são completamente diferentes das variáveis locais de outra função.

Variáveis globais podem ser lidas a partir de um escopo local

Considere o programa a seguir:

```
def spam():
    print(eggs)
eggs = 42
spam()
print(eggs)
```

Como não há nenhum parâmetro chamado eggs nem qualquer código que atribua um valor a eggs na função spam(), quando eggs é usada nessa função, o

Python a considera como uma referência à variável global eggs. É por isso que 42 é exibido quando o programa anterior é executado.

Variáveis locais e globais com o mesmo nome

Para simplificar sua vida, evite usar variáveis locais que tenham o mesmo nome que uma variável global ou outra variável local. Porém, tecnicamente, é perfeitamente permitido fazer isso em Python. Para ver o que acontece, digite o código a seguir no editor de arquivo e salve-o como *sameName.py*:

```
  def spam():
u     eggs = 'spam local'
      print(eggs)    # exibe 'spam local'

  def bacon():
v     eggs = 'bacon local'
      print(eggs)    # exibe 'bacon local'
      spam()
      print(eggs)    # exibe 'bacon local'

w eggs = 'global'
  bacon()
  print(eggs)        # exibe 'global'
```

Ao executar esse programa, sua saída será:

```
bacon local
spam local
bacon local
global
```

Na realidade, há três variáveis diferentes nesse programa, porém, de modo confuso, todas elas foram chamadas de eggs. As variáveis são as seguintes:

u Uma variável chamada eggs existente em um escopo local quando spam() é chamada.

v Uma variável chamada eggs existente em um escopo local quando bacon() é chamada.

w Uma variável chamada eggs existente no escopo global.

Como essas três variáveis diferentes têm o mesmo nome, pode ser confuso

perceber qual delas está sendo usada em determinado instante. É por isso que devemos evitar o uso do mesmo nome de variável em escopos diferentes.

Instrução global

Caso seja necessário modificar uma variável global em uma função, utilize a instrução global. Se você tiver uma linha como global eggs no início de uma função, ela dirá ao Python que "nesta função, eggs refere-se à variável global, portanto não crie uma variável local com esse nome". Por exemplo, digite o código a seguir no editor de arquivo e salve-o como *sameName2.py*:

```
  def spam():
u     global eggs
v     eggs = 'spam'

  eggs = 'global'
  spam()
  print(eggs)
```

Ao executar esse programa, a chamada a print() no final exibirá:

```
spam
```

Como eggs é declarada como global no início de spam() u, quando eggs é definida com 'spam' v, essa atribuição é feita ao eggs do escopo global. Nenhuma variável local eggs será criada.

Há quatro regras para dizer se uma variável está em um escopo local ou global:

1. Se uma variável estiver sendo usada no escopo global (ou seja, fora de todas as funções), ela sempre será uma variável global.
2. Se houver uma instrução global para essa variável em uma função, ela será uma variável global.
3. Caso contrário, se a variável for usada em uma instrução de atribuição na função, ela será uma variável local.
4. Porém, se a variável não for usada em uma instrução de atribuição, ela será uma variável global.

Para ter uma melhor noção dessas regras, a seguir apresentamos um

programa de exemplo. Digite o código a seguir no editor de arquivo e salve-o como *sameName3.py*:

```
  def spam():
u     global eggs
      eggs = 'spam' # essa é a variável global
  def bacon():
v     eggs = 'bacon' # essa é uma variável local

  def ham():
w     print(eggs) # essa é a variável global

  eggs = 42 # essa é a variável global
  spam()
  print(eggs)
```

Na função spam(), eggs é a variável global eggs, pois há uma instrução global para eggs no início da função u. Em bacon(), eggs é uma variável local, pois há uma instrução de atribuição para ela nessa função v. Em ham() w, eggs é a variável global, pois não há nenhuma instrução de atribuição nem instrução global para ela nessa função. Se *sameName3.py* for executado, a saída terá o seguinte aspecto:

```
spam
```

Em uma função, uma variável será sempre global ou sempre local. Não há nenhuma maneira de o código de uma função poder usar uma variável local chamada eggs e, mais adiante nessa mesma função, usar a variável global eggs.

NOTA Se quiser modificar o valor armazenado em uma variável global em uma função, utilize uma instrução global nessa variável.

Se você tentar usar uma variável local em uma função antes de atribuir um valor a ela, como no programa a seguir, o Python apresentará um erro. Para ver isso, digite o código a seguir no editor de arquivo e salve-o como *sameName4.py*:

```
  def spam():
      print(eggs) # ERRO!
u     eggs = 'spam local'
```

```
v eggs = 'global'
  spam()
```

Se você executar o programa anterior, uma mensagem de erro será gerada.

```
Traceback (most recent call last):
  File "C:/test3784.py", line 6, in <module>
    spam()
  File "C:/test3784.py", line 2, in spam
    print(eggs) # ERROR!
UnboundLocalError: local variable 'eggs' referenced before assignment
```

Esse erro ocorre porque o Python percebe que há uma instrução de atribuição para eggs na função spam() u e, desse modo, considera eggs como sendo local. Entretanto, como print(eggs) é executado antes de eggs receber qualquer valor, a variável local eggs não existe. O Python *não* optará por usar a variável global eggs v.

FUNÇÕES COMO "CAIXAS-PRETAS"

Com frequência, tudo que devemos saber sobre uma função são suas entradas (os parâmetros) e o valor de saída; nem sempre será necessário se dar o trabalho de saber como o código da função realmente opera. Quando pensamos nas funções dessa maneira, em nível mais geral, é comum dizer que estamos tratando a função como uma "caixa-preta".

Essa ideia é fundamental na programação moderna. Capítulos mais adiante neste livro mostrarão diversos módulos com funções criadas por outras pessoas. Embora você possa dar uma espiada no código-fonte se estiver curioso, não será necessário conhecer o seu funcionamento para poder usá-lo. Pelo fato de escrever funções sem variáveis globais ser incentivado, geralmente, não será preciso se preocupar com a interação do código da função com o restante de seu programa.

Tratamento de exceções

Neste momento, obter um erro, isto é, uma *exceção* em seu programa Python quer dizer que o programa como um todo falhará. Você não vai querer que isso aconteça em programas do mundo real. Em vez disso, queremos que o programa detecte erros, trate-os e então continue a executar.

Por exemplo, considere o programa a seguir, que contém um erro de

“divisão por zero”. Abra uma nova janela no editor de arquivo e insira o código a seguir, salvando-o como *zeroDivide.py*:

```
def spam(divideBy):
    return 42 / divideBy

print(spam(2))
print(spam(12))
print(spam(0))
print(spam(1))
```

Definimos uma função chamada spam, fornecemos um parâmetro a ela e, em seguida, exibimos o valor dessa função com diversos parâmetros para ver o que acontece. Essa é a saída que você obterá quando executar o código anterior:

```
21.0
3.5
Traceback (most recent call last):
  File "C:/zeroDivide.py", line 6, in <module>
    print(spam(0))
  File "C:/zeroDivide.py", line 2, in spam
    return 42 / divideBy
ZeroDivisionError: division by zero
```

Um ZeroDivisionError ocorre sempre que você tenta dividir um número por zero. A partir do número da linha apresentado na mensagem de erro, sabemos que a instrução return em spam() está provocando um erro.

Os erros podem ser tratados com instruções try e except. O código que pode conter um erro em potencial é inserido em uma cláusula try. A execução do programa segue para o início da cláusula except seguinte caso um erro ocorra.

Podemos colocar o código anterior de divisão por zero em uma cláusula try e fazer com que uma cláusula except contenha um código para lidar com o que acontece quando esse erro ocorrer.

```
def spam(divideBy):
    try:
        return 42 / divideBy
    except ZeroDivisionError:
        print('Error: Invalid argument.')
```

```
print(spam(2))
print(spam(12))
print(spam(0))
print(spam(1))
```

Quando um código em uma cláusula try provocar um erro, a execução do programa será transferida imediatamente para o código na cláusula except. Após executar esse código, a execução continuará normalmente. A saída do programa anterior será:

```
21.0
3.5
Error: Invalid argument.
None
42.0
```

Observe que qualquer erro ocorrido em chamadas de função em um bloco try também será capturado. Considere o programa a seguir, que apresenta as chamadas a spam() em um bloco try:

```
def spam(divideBy):
    return 42 / divideBy

try:
    print(spam(2))
    print(spam(12))
    print(spam(0))
    print(spam(1))
except ZeroDivisionError:
    print('Error: Invalid argument.')
```

Quando esse programa for executado, a saída terá o seguinte aspecto:

```
21.0
3.5
Error: Invalid argument.
```

print(spam(1)) não é executado porque, depois que a execução passa para o código da cláusula except, ela não retorna à cláusula try. Em vez disso, ela continua se movendo normalmente para baixo.

Um pequeno programa: adivinhe o número

Os exemplos lúdicos que mostrei até agora são úteis para apresentar os conceitos básicos, porém vamos agora ver como tudo que aprendemos pode ser reunido em um programa mais completo. Nesta seção, mostrarei um jogo simples de "adivinhar o número". Ao executar esse programa, a saída terá uma aparência semelhante a:

```
I am thinking of a number between 1 and 20.
Take a guess.
10
Your guess is too low.
Take a guess.
15
Your guess is too low.
Take a guess.
17
Your guess is too high.
Take a guess.
16
Good job! You guessed my number in 4 guesses!
```

Digite o código-fonte a seguir no editor de arquivo e salve-o como *guessTheNumber.py*:

```
# Este é um jogo de adivinhar o número.
import random
secretNumber = random.randint(1, 20)
print('I am thinking of a number between 1 and 20.')

# Peça para o jogador adivinhar 6 vezes.
for guessesTaken in range(1, 7):
    print('Take a guess.')
    guess = int(input())

    if guess < secretNumber:
        print('Your guess is too low.')
    elif guess > secretNumber:
        print('Your guess is too high.')
    else:
        break    # Esta condição corresponde ao palpite correto!
```

```
if guess == secretNumber:
    print('Good job! You guessed my number in ' + str(guessesTaken) + ' guesses!')
else:
    print('Nope. The number I was thinking of was ' + str(secretNumber))
```

Vamos analisar esse código linha a linha, começando do início.

```
# Este é um jogo de adivinhar o número.
import random
secretNumber = random.randint(1, 20)
```

Inicialmente, um comentário no início do programa explica o que ele faz. Em seguida, o programa importa o módulo random para que seja possível utilizar a função random.randint() e gerar um número para o usuário adivinhar. O valor de retorno, que é um inteiro aleatório entre 1 e 20, é armazenado na variável secretNumber.

```
print('I am thinking of a number between 1 and 20.')

# Peça para o jogador adivinhar 6 vezes.
for guessesTaken in range(1, 7):
    print('Take a guess.')
    guess = int(input())
```

O programa informa o jogador que tem um número secreto e que dará seis chances a ele para adivinhá-lo. O código que permite que o jogador forneça um palpite e verifica-o está em um loop for que executará no máximo seis vezes. A primeira ação que ocorre no loop é o jogador digitar um palpite. Como input() retorna uma string, seu valor de retorno é passado diretamente a int(), que traduz a string para um valor inteiro. Esse valor é armazenado em uma variável chamada guess.

```
    if guess < secretNumber:
        print('Your guess is too low.')
    elif guess > secretNumber:
        print('Your guess is too high.')
```

Essas linhas de código verificam se o palpite é menor ou maior que o número secreto. Qualquer que seja o caso, uma dica será exibida na tela.

```
    else:
        break    # Esta condição corresponde ao palpite correto!
```

Se o palpite não for maior nem menor que o número secreto, ele deverá ser igual a esse número, caso em que você vai querer que a execução do programa saia do loop for.

```
if guess == secretNumber:
    print('Good job! You guessed my number in ' + str(guessesTaken) + ' guesses!')
else:
    print('Nope. The number I was thinking of was ' + str(secretNumber))
```

Depois do loop for, a instrução if...else anterior verifica se o jogador adivinhou corretamente o número e exibe uma mensagem apropriada na tela. Em ambos os casos, o programa exibe uma variável que contém um valor inteiro (guessesTaken e secretNumber). Como esses valores inteiros devem ser concatenados em strings, essas variáveis são passadas à função str(), que retorna o valor em forma de string desses inteiros. Agora essas strings podem ser concatenadas com os operadores + antes de, finalmente, serem passadas para a chamada da função print().

Resumo

As funções são a maneira principal de compartimentar o seu código em grupos lógicos. Como as variáveis nas funções existem em seus próprios escopos locais, o código de uma função não pode afetar diretamente os valores de variáveis em outras funções. Isso limita os códigos que podem alterar os valores de suas variáveis, o que pode ser útil quando se trata de fazer debugging de seu código.

As funções são uma ótima ferramenta para ajudar a organizar o seu código. Podemos pensar nelas como caixas-pretas: elas recebem dados de entrada na forma de parâmetros e têm saídas na forma de valores de retorno; além disso, seu código não afeta as variáveis de outras funções.

Nos capítulos anteriores, um único erro poderia causar uma falha em seus programas. Neste capítulo, conhecemos as instruções try e except, que podem executar códigos quando um erro for detectado. Elas podem tornar seus

programas mais resilientes a casos de erros comuns.

Exercícios práticos

1. Por que é vantajoso ter funções em seus programas?
2. Em que momento o código de uma função é executado: quando a função é definida ou quando ela é chamada?
3. Que instrução cria uma função?
4. Qual é a diferença entre uma função e uma chamada de função?
5. Quantos escopos globais existem em um programa Python? Quantos escopos locais?
6. O que acontece às variáveis em um escopo local quando a chamada da função retorna?
7. O que é um valor de retorno? Um valor de retorno pode fazer parte de uma expressão?
8. Se uma função não tiver uma instrução de retorno, qual será o valor de retorno de uma chamada a essa função?
9. Como podemos fazer com que uma variável em uma função refira-se à variável global?
10. Qual é o tipo de dado de None?
11. O que a instrução import areallyourpetsnamederic faz?
12. Se você tivesse uma função chamada bacon() em um módulo chamado spam, como você a chamaria após ter importado spam?
13. Como podemos evitar que um programa falhe quando houver um erro?
14. De que é composta a cláusula try? De que é composta a cláusula except?

Projetos práticos

Para exercitar, escreva programas que executem as tarefas a seguir.

Sequência de Collatz

Crie uma função chamada collatz() que tenha um parâmetro de nome number. Se number for par, collatz() deverá exibir number // 2 e retornar esse valor. Se

number for ímpar, collatz() deverá exibir e retornar 3 * number + 1.

Em seguida, crie um programa que permita que o usuário digite um inteiro e fique chamando collatz() com esse número até a função retornar o valor 1. (De modo bastante surpreendente, essa sequência, na realidade, funciona para qualquer inteiro – cedo ou tarde, ao usar essa sequência, você chegará em 1! Até mesmo os matemáticos não têm muita certeza do motivo. Seu programa está explorando o que chamamos de *sequência de Collatz*, às vezes chamada de "o mais simples problema matemático impossível".)

Lembre-se de converter o valor de retorno de input() em um inteiro usando a função int(); caso contrário, o valor será do tipo string.

Dica: um number inteiro será par se number % 2 == 0 e ímpar se number % 2 == 1.

A saída desse programa poderá ter uma aparência semelhante a:

```
Enter number:
3
10
5
16
8
4
2
1
```

Validação de dados de entrada

Acrescente instruções try e except no projeto anterior para detectar se o usuário digitou uma string que não corresponda a um inteiro. Normalmente, a função int() gerará um erro ValueError se receber uma string que não seja um inteiro, como em int('puppy'). Na cláusula except, exiba uma mensagem ao usuário informando-lhe que um inteiro deve ser fornecido.

CAPÍTULO 4

LISTAS

Outro assunto que você deverá entender antes de poder realmente começar a criar programas é o tipo de dado lista (list) e sua prima, a tupla (tuple). As listas e as tuplas podem conter diversos valores, o que facilita criar programas que lidem com grandes quantidades de dados. Como as listas podem conter outras listas, podemos usá-las para organizar dados em estruturas hierárquicas.

Neste capítulo, discutirei o básico sobre as listas. Também falarei sobre os métodos, que são funções associadas a valores de determinados tipos de dados. Em seguida, discutirei brevemente os tipos de dados tupla e string, que são semelhantes à lista, além de compará-los aos valores de lista. No próximo capítulo, apresentarei o tipo de dado dicionário (dictionary).

Tipo de dado lista

Uma *lista* é um valor que contém diversos valores em uma sequência ordenada. O termo *valor de lista* refere-se à lista propriamente dita (que é um valor que pode ser armazenado em uma variável ou passado para uma função, como qualquer outro valor), e não aos valores contidos no valor da lista. Um valor de lista tem o seguinte aspecto: ['cat', 'bat', 'rat', 'elephant']. Assim como os valores de string são digitados com aspas para marcar em que ponto a string começa e termina, uma lista começa com um colchete de abertura e termina com um colchete de fechamento, ou seja, []. Os valores contidos na lista também são chamados de *itens*. Os itens são separados por vírgulas (ou seja, são *delimitados por vírgula*). Por exemplo, digite o seguinte no shell interativo:

```
>>> [1, 2, 3]
[1, 2, 3]
>>> ['cat', 'bat', 'rat', 'elephant']
['cat', 'bat', 'rat', 'elephant']
>>> ['hello', 3.1415, True, None, 42]
```

```
['hello', 3.1415, True, None, 42]
u >>> spam = ['cat', 'bat', 'rat', 'elephant']
>>> spam
['cat', 'bat', 'rat', 'elephant']
```

A variável spam u continua recebendo um único valor: o valor da lista. Porém o valor da lista em si contém outros valores. O valor [] corresponde a uma lista vazia, sem valores, semelhante a '', que é a string vazia.

Obtendo valores individuais de uma lista por meio de índices

Suponha que você tenha a lista ['cat', 'bat', 'rat', 'elephant'] armazenada em uma variável chamada spam. O código Python spam[0] será avaliado como 'cat', spam[1] será avaliado como 'bat' e assim por diante. O inteiro entre os colchetes após o nome da lista chama-se *índice*. O primeiro valor da lista está no índice 0, o segundo valor está no índice 1, o terceiro valor está no índice 2 e assim por diante. A figura 4.1 mostra um valor de lista atribuído a spam, juntamente com os valores para os quais as expressões com índice são avaliadas.

Figura 4.1 – Um valor de lista armazenado na variável **spam***, mostrando o valor ao qual cada índice se refere.*

Por exemplo, digite as expressões a seguir no shell interativo. Comece atribuindo uma lista à variável spam.

```
>>> spam = ['cat', 'bat', 'rat', 'elephant']
>>> spam[0]
'cat'
>>> spam[1]
'bat'
>>> spam[2]
'rat'
>>> spam[3]
'elephant'
```

```
>>> ['cat', 'bat', 'rat', 'elephant'][3]
'elephant'
u >>> 'Hello ' + spam[0]
v 'Hello cat'
>>> 'The ' + spam[1] + ' ate the ' + spam[0] + '.'
'The bat ate the cat.'
```

Observe que a expressão 'Hello ' + spam[0] u é avaliada como 'Hello ' + 'cat' porque spam[0] é avaliada como a string 'cat'. Essa expressão, por sua vez, é avaliada como o valor de string 'Hello cat' v.

O Python apresentará uma mensagem de erro IndexError se você utilizar um índice que exceda a quantidade de valores de seu valor de lista.

```
>>> spam = ['cat', 'bat', 'rat', 'elephant']
>>> spam[10000]
Traceback (most recent call last):
  File "<pyshell#9>", line 1, in <module>
    spam[10000]
IndexError: list index out of range
```

Os índices podem ser somente valores inteiros; não podem ser números de ponto flutuante. O exemplo a seguir provoca um erro TypeError:

```
>>> spam = ['cat', 'bat', 'rat', 'elephant']
>>> spam[1]
'bat'
>>> spam[1.0]
Traceback (most recent call last):
  File "<pyshell#13>", line 1, in <module>
    spam[1.0]
TypeError: list indices must be integers, not float
>>> spam[int(1.0)]
'bat'
```

As listas também podem conter outros valores de lista. Os valores dessas listas de listas podem ser acessados usando índices múltiplos, da seguinte maneira:

```
>>> spam = [['cat', 'bat'], [10, 20, 30, 40, 50]]
>>> spam[0]
['cat', 'bat']
```

```
>>> spam[0][1]
'bat'
>>> spam[1][4]
50
```

O primeiro índice determina o valor de lista a ser usado e o segundo indica o valor dentro do valor de lista. Por exemplo, spam[0][1] exibe 'bat', que é o segundo valor da primeira lista. Se apenas um índice for utilizado, o programa exibirá o valor de lista completo que estiver nesse índice.

Índices negativos

Embora os índices comecem em 0 e aumentem, também podemos usar inteiros negativos para o índice. O valor inteiro -1 refere-se ao último índice de uma lista, o valor -2 refere-se ao penúltimo índice de uma lista e assim por diante. Digite o seguinte no shell interativo:

```
>>> spam = ['cat', 'bat', 'rat', 'elephant']
>>> spam[-1]
'elephant'
>>> spam[-3]
'bat'
>>> 'The ' + spam[-1] + ' is afraid of the ' + spam[-3] + '.'
'The elephant is afraid of the bat.'
```

Obtendo sublistas com slices

Assim como um índice pode acessar um único valor de uma lista, um *slice* (fatia) pode obter diversos valores de uma lista na forma de uma nova lista. Um slice é digitado entre colchetes, como um índice, porém tem dois inteiros separados por dois-pontos. Observe a diferença entre índices e slices.

- spam[2] é uma lista com um índice (um inteiro).
- spam[1:4] é uma lista com um slice (dois inteiros).

Em um slice, o primeiro inteiro é o índice em que o slice começa. O segundo inteiro é o índice em que o slice termina. Um slice vai até o valor do segundo índice, sem incluí-lo, e é avaliado como um novo valor de lista. Digite o seguinte no shell interativo:

```
>>> spam = ['cat', 'bat', 'rat', 'elephant']
```

```
>>> spam[0:4]
['cat', 'bat', 'rat', 'elephant']
>>> spam[1:3]
['bat', 'rat']
>>> spam[0:-1]
['cat', 'bat', 'rat']
```

Como atalho, podemos deixar de especificar um dos índices de cada lado dos dois-pontos do slice ou ambos. Deixar de especificar o primeiro índice é o mesmo que usar 0, ou seja, o início da lista. Deixar de especificar o segundo índice é o mesmo que usar o tamanho da lista, e o slice terminará no final da lista. Digite o seguinte no shell interativo:

```
>>> spam = ['cat', 'bat', 'rat', 'elephant']
>>> spam[:2]
['cat', 'bat']
>>> spam[1:]
['bat', 'rat', 'elephant']
>>> spam[:]
['cat', 'bat', 'rat', 'elephant']
```

Obtendo o tamanho de uma lista com len()

A função len() retorna a quantidade de valores presentes em um valor de lista passado para ela, do mesmo modo que ela pode contabilizar o número de caracteres em um valor do tipo string. Digite o seguinte no shell interativo:

```
>>> spam = ['cat', 'dog', 'moose']
>>> len(spam)
3
```

Alterando valores de uma lista usando índices

Normalmente, um nome de variável é colocado do lado esquerdo de uma instrução de atribuição, como em spam = 42. Entretanto também podemos utilizar um índice de uma lista para mudar o valor presente nesse índice. Por exemplo, spam[1] = 'aardvark' quer dizer "atribua a string 'aardvark' ao valor no índice 1 da lista spam". Digite o seguinte no shell interativo:

```
>>> spam = ['cat', 'bat', 'rat', 'elephant']
```

```
>>> spam[1] = 'aardvark'
>>> spam
['cat', 'aardvark', 'rat', 'elephant']
>>> spam[2] = spam[1]
>>> spam
['cat', 'aardvark', 'aardvark', 'elephant']
>>> spam[-1] = 12345
>>> spam
['cat', 'aardvark', 'aardvark', 12345]
```

Concatenação e repetição de listas

O operador + pode combinar duas listas para criar um novo valor de lista, da mesma maneira que combina duas strings gerando um novo valor de string. O operador + também pode ser usado com uma lista e um valor inteiro para repetir a lista. Digite o seguinte no shell interativo:

```
>>> [1, 2, 3] + ['A', 'B', 'C']
[1, 2, 3, 'A', 'B', 'C']
>>> ['X', 'Y', 'Z'] * 3
['X', 'Y', 'Z', 'X', 'Y', 'Z', 'X', 'Y', 'Z']
>>> spam = [1, 2, 3]
>>> spam = spam + ['A', 'B', 'C']
>>> spam
[1, 2, 3, 'A', 'B', 'C']
```

Removendo valores de listas usando instruções del

A instrução del apagará valores de um índice em uma lista. Todos os valores da lista após o valor ter sido apagado serão deslocados em um índice. Por exemplo, digite o seguinte no shell interativo:

```
>>> spam = ['cat', 'bat', 'rat', 'elephant']
>>> del spam[2]
>>> spam
['cat', 'bat', 'elephant']
>>> del spam[2]
>>> spam
['cat', 'bat']
```

A instrução del também pode ser usada em uma variável simples para

apagá-la, como se fosse o "inverso" de uma instrução de atribuição. Se você tentar utilizar a variável após removê-la, um erro NameError será obtido, pois a variável não existe mais.

Na prática, raramente será necessário apagar variáveis simples. A instrução del é usada principalmente para apagar valores de listas.

Trabalhando com listas

Ao começar a escrever programas, é tentador criar diversas variáveis individuais para armazenar um grupo de valores semelhantes. Por exemplo, se eu quisesse armazenar os nomes de meus gatos, poderia me sentir tentado a criar um código como:

```
catName1 = 'Zophie'
catName2 = 'Pooka'
catName3 = 'Simon'
catName4 = 'Lady Macbeth'
catName5 = 'Fat-tail'
catName6 = 'Miss Cleo'
```

(Não tenho tantos gatos assim, eu juro.) Acontece que essa é uma maneira ruim de escrever um código. Para começar, se a quantidade de gatos mudar, seu programa jamais poderá armazenar mais gatos do que a quantidade de variáveis que você tiver. Esses tipos de programa também têm muito código duplicado ou quase idêntico. Considere a quantidade de código duplicado presente no programa a seguir, que você deve digitar no editor de arquivo e salvar como *allMyCats1.py*:

```
print('Enter the name of cat 1:')
catName1 = input()
print('Enter the name of cat 2:')
catName2 = input()
print('Enter the name of cat 3:')
catName3 = input()
print('Enter the name of cat 4:')
catName4 = input()
print('Enter the name of cat 5:')
catName5 = input()
print('Enter the name of cat 6:')
```

```
catName6 = input()
print('The cat names are:')
print(catName1 + ' ' + catName2 + ' ' + catName3 + ' ' + catName4 + ' ' + catName5 + ' '
+ catName6)
```

Em vez de usar diversas variáveis repetitivas, podemos usar uma única variável que contenha um valor de lista. Por exemplo, a seguir, apresentamos uma versão nova e melhorada do programa *allMyCats1.py*. Essa nova versão utiliza uma única lista e é capaz de armazenar qualquer quantidade de gatos que o usuário digitar. Em uma nova janela do editor de arquivo, digite o código-fonte a seguir e salve-o como *allMyCats2.py*:

```
catNames = []
while True:
    print('Enter the name of cat ' + str(len(catNames) + 1) +
      ' (Or enter nothing to stop.):')
    name = input()
    if name == '':
        break
    catNames = catNames + [name]  # concatenação de lista
print('The cat names are:')
for name in catNames:
    print('  ' + name)
```

Ao executar esse programa, a saída terá uma aparência semelhante a:

```
Enter the name of cat 1 (Or enter nothing to stop.):
Zophie
Enter the name of cat 2 (Or enter nothing to stop.):
Pooka
Enter the name of cat 3 (Or enter nothing to stop.):
Simon
Enter the name of cat 4 (Or enter nothing to stop.):
Lady Macbeth
Enter the name of cat 5 (Or enter nothing to stop.):
Fat-tail
Enter the name of cat 6 (Or enter nothing to stop.):
Miss Cleo
Enter the name of cat 7 (Or enter nothing to stop.):

The cat names are:
```

```
Zophie
Pooka
Simon
Lady Macbeth
Fat-tail
Miss Cleo
```

A vantagem de usar uma lista é que agora seus dados estão em uma estrutura; sendo assim, seu programa será muito mais flexível para processar os dados do que seria se tivesse diversas variáveis repetitivas.

Utilizando loops for com listas

No capítulo 2, aprendemos a usar loops `for` para executar um bloco de código um número específico de vezes. Tecnicamente, um loop `for` repete o bloco de código uma vez para cada valor de uma lista ou de um valor semelhante a uma lista. Por exemplo, se você executar o código a seguir

```
for i in range(4):
    print(i)
```

a saída desse programa será:

```
0
1
2
3
```

Isso ocorre porque o valor de retorno de `range(4)` é um valor semelhante a uma lista, que o Python considera como `[0, 1, 2, 3]`. O programa a seguir apresenta a mesma saída que o programa anterior:

```
for i in [0, 1, 2, 3]:
    print(i)
```

O que o loop `for` anterior faz é executar sua cláusula com a variável `i` definida com um valor sucessivo da lista `[0, 1, 2, 3]` a cada iteração.

NOTA Neste livro, utilizo o termo *semelhante à lista* para referir-me aos tipos de dados que, tecnicamente, são chamados de *sequências*. Contudo não é preciso conhecer as definições técnicas desse termo.

Uma técnica Python comum consiste em usar `range(len(`*algumaLista*`))` com um

loop for para fazer uma iteração pelos índices de uma lista. Por exemplo, digite o seguinte no shell interativo:

```
>>> supplies = ['pens', 'staplers', 'flame-throwers', 'binders']
>>> for i in range(len(supplies)):
    print('Index ' + str(i) + ' in supplies is: ' + supplies[i])

Index 0 in supplies is: pens
Index 1 in supplies is: staplers
Index 2 in supplies is: flame-throwers
Index 3 in supplies is: binders
```

Usar range(len(supplies)) no loop for mostrado anteriormente é prático porque o código do loop pode acessar o índice (como a variável i) e o valor nesse índice (como supplies[i]). Melhor ainda é o fato de range(len(supplies)) fazer a iteração por todos os índices de supplies, independentemente da quantidade de itens contidos na lista.

Operadores in e not in

Podemos determinar se um valor está ou não em uma lista usando os operadores in e not in. Assim como os demais operadores, in e not in são usados em expressões e associam dois valores: um valor a ser procurado em uma lista e a lista em que esse valor poderá ser encontrado. Essas expressões serão avaliadas como um valor booleano. Digite o seguinte no shell interativo:

```
>>> 'howdy' in ['hello', 'hi', 'howdy', 'heyas']
True
>>> spam = ['hello', 'hi', 'howdy', 'heyas']
>>> 'cat' in spam
False
>>> 'howdy' not in spam
False
>>> 'cat' not in spam
True
```

Por exemplo, o programa a seguir permite que o usuário digite o nome de um bichinho de estimação e, em seguida, verifique se o nome está em uma lista de bichinhos de estimação. Abra uma nova janela no editor de arquivo,

insira o código a seguir e salve-o como *myPets.py*:

```
myPets = ['Zophie', 'Pooka', 'Fat-tail']
print('Enter a pet name:')
name = input()
if name not in myPets:
    print('I do not have a pet named ' + name)
else:
    print(name + ' is my pet.')
```

A saída poderá ter um aspecto semelhante a:

```
Enter a pet name:
Footfoot
I do not have a pet named Footfoot
```

Truque da atribuição múltipla

O *truque da atribuição múltipla* é um atalho que permite atribuir os valores de uma lista a diversas variáveis em uma única linha de código. Desse modo, em vez de implementar

```
>>> cat = ['fat', 'black', 'loud']
>>> size = cat[0]
>>> color = cat[1]
>>> disposition = cat[2]
```

você pode digitar a linha de código a seguir:

```
>>> cat = ['fat', 'black', 'loud']
>>> size, color, disposition = cat
```

A quantidade de variáveis e o tamanho da lista devem ser exatamente iguais; caso contrário, o Python apresentará um ValueError:

```
>>> cat = ['fat', 'black', 'loud']
>>> size, color, disposition, name = cat
Traceback (most recent call last):
  File "<pyshell#84>", line 1, in <module>
    size, color, disposition, name = cat
ValueError: need more than 3 values to unpack
```

Operadores de atribuição expandidos

Quando atribuímos um valor a uma variável, com frequência, utilizamos a própria variável. Por exemplo, após atribuir 42 à variável spam, podemos incrementar o valor de spam em 1 com o código a seguir:

```
>>> spam = 42
>>> spam = spam + 1
>>> spam
43
```

Como atalho, podemos usar o operador de atribuição expandido += para fazer o mesmo:

```
>>> spam = 42
>>> spam += 1
>>> spam
43
```

Há operadores de atribuição expandidos para os operadores +, -, *, / e %, descritos na tabela 4.1.

Tabela 4.1 – Os operadores de atribuição expandidos

Instrução de atribuição expandida	Instrução de atribuição equivalente
spam += 1	spam = spam + 1
spam -= 1	spam = spam - 1
spam *= 1	spam = spam * 1
spam /= 1	spam = spam / 1
spam %= 1	spam = spam % 1

O operador += também pode realizar concatenação de strings e de listas e o operador *= pode fazer repetição de string e de lista. Digite o seguinte no shell interativo:

```
>>> spam = 'Hello'
>>> spam += ' world!'
>>> spam
'Hello world!'
>>> bacon = ['Zophie']
>>> bacon *= 3
```

```
>>> bacon
['Zophie', 'Zophie', 'Zophie']
```

Métodos

Um *método* é o mesmo que uma função, exceto pelo fato de ser chamado "sobre um valor". Por exemplo, se um valor de lista estiver armazenado em spam, podemos chamar o método de lista index() (que explicarei a seguir) nessa lista, da seguinte maneira: spam.index('hello'). A parte referente ao método é inserida depois do valor, separada por um ponto.

Cada tipo de dado tem seu próprio conjunto de métodos. O tipo de dado lista, por exemplo, tem diversos métodos úteis para encontrar, adicionar, remover e manipular valores em uma lista.

Encontrando um valor em uma lista com o método index()

Os valores de lista têm um método index() que pode receber um valor e, se esse valor estiver presente na lista, seu índice será retornado. Se o valor não estiver presente na lista, o Python apresentará um erro ValueError. Digite o seguinte no shell interativo:

```
>>> spam = ['hello', 'hi', 'howdy', 'heyas']
>>> spam.index('hello')
0
>>> spam.index('heyas')
3
>>> spam.index('howdy howdy howdy')
Traceback (most recent call last):
  File "<pyshell#31>", line 1, in <module>
    spam.index('howdy howdy howdy')
ValueError: 'howdy howdy howdy' is not in list
```

Quando houver duplicatas de valores na lista, o índice da primeira ocorrência será retornado. Digite o seguinte no shell interativo e observe que index() retorna 1, e não 3:

```
>>> spam = ['Zophie', 'Pooka', 'Fat-tail', 'Pooka']
>>> spam.index('Pooka')
```

Adicionando valores a listas com os métodos append() e insert()

Para adicionar novos valores a uma lista, utilize os métodos append() e insert(). Digite o seguinte no shell interativo para chamar o método append() em um valor de lista armazenado na variável spam:

```
>>> spam = ['cat', 'dog', 'bat']
>>> spam.append('moose')
>>> spam
['cat', 'dog', 'bat', 'moose']
```

A chamada anterior ao método append() adiciona o argumento no final da lista. O método insert() pode inserir um valor em qualquer índice da lista. O primeiro argumento de insert() é o índice do novo valor e o segundo argumento é o novo valor a ser inserido. Digite o seguinte no shell interativo:

```
>>> spam = ['cat', 'dog', 'bat']
>>> spam.insert(1, 'chicken')
>>> spam
['cat', 'chicken', 'dog', 'bat']
```

Observe que o código é spam.append('moose') e spam.insert(1, 'chicken'), e não spam = spam.append('moose') e spam = spam.insert(1, 'chicken'). Nem append() nem insert() retornam o novo valor de spam como seu valor de retorno. (De fato, o valor de retorno de append() e de insert() é None, portanto, definitivamente, você não vai querer armazenar esse valor como o novo valor da variável.) Em vez disso, a lista é modificada *in place* (no próprio local). A modificação de uma lista in place será discutida com mais detalhes mais adiante na seção “Tipos de dados mutáveis e imutáveis”.

Os métodos pertencem a um único tipo de dado. Os métodos append() e insert() são métodos de lista e podem ser chamados somente sobre valores do tipo lista, e não em outros valores como strings ou inteiros. Digite o seguinte no shell interativo e observe as mensagens de erro AttributeError geradas:

```
>>> eggs = 'hello'
>>> eggs.append('world')
```

```
Traceback (most recent call last):
  File "<pyshell#19>", line 1, in <module>
    eggs.append('world')
AttributeError: 'str' object has no attribute 'append'
>>> bacon = 42
>>> bacon.insert(1, 'world')
Traceback (most recent call last):
  File "<pyshell#22>", line 1, in <module>
    bacon.insert(1, 'world')
AttributeError: 'int' object has no attribute 'insert'
```

Removendo valores de listas com remove()

O método remove() recebe o valor a ser removido da lista em que é chamada. Digite o seguinte no shell interativo:

```
>>> spam = ['cat', 'bat', 'rat', 'elephant']
>>> spam.remove('bat')
>>> spam
['cat', 'rat', 'elephant']
```

A tentativa de apagar um valor que não exista na lista resultará em um erro ValueError. Por exemplo, digite o seguinte no shell interativo e observe o erro apresentado:

```
>>> spam = ['cat', 'bat', 'rat', 'elephant']
>>> spam.remove('chicken')
Traceback (most recent call last):
  File "<pyshell#11>", line 1, in <module>
    spam.remove('chicken')
ValueError: list.remove(x): x not in list
```

Se o valor aparecer diversas vezes na lista, somente a primeira ocorrência desse valor será removida. Digite o seguinte no shell interativo:

```
>>> spam = ['cat', 'bat', 'rat', 'cat', 'hat', 'cat']
>>> spam.remove('cat')
>>> spam
['bat', 'rat', 'cat', 'hat', 'cat']
```

A instrução del é apropriada quando conhecemos o índice do valor que queremos remover da lista. O método remove() é adequado quando

conhecemos o valor que queremos remover da lista.

Ordenando os valores de uma lista com o método sort()

As listas de valores numéricos ou de strings podem ser ordenadas com o método sort(). Por exemplo, digite o seguinte no shell interativo:

```
>>> spam = [2, 5, 3.14, 1, -7]
>>> spam.sort()
>>> spam
[-7, 1, 2, 3.14, 5]
>>> spam = ['ants', 'cats', 'dogs', 'badgers', 'elephants']
>>> spam.sort()
>>> spam
['ants', 'badgers', 'cats', 'dogs', 'elephants']
```

Podemos também passar True para o argumento nomeado reverse de modo a fazer com que sort() ordene os valores em ordem inversa. Digite o seguinte no shell interativo:

```
>>> spam.sort(reverse=True)
>>> spam
['elephants', 'dogs', 'cats', 'badgers', 'ants']
```

Há três aspectos que você deve observar em relação ao método sort(). Em primeiro lugar, o método sort() ordena a lista in place; não tente capturar o valor de retorno escrevendo um código como spam = spam.sort().

Em segundo lugar, não podemos ordenar listas que contenham valores numéricos *e* valores do tipo string, pois o Python não saberá como comparar esses valores. Digite o seguinte no shell interativo e observe o erro TypeError:

```
>>> spam = [1, 3, 2, 4, 'Alice', 'Bob']
>>> spam.sort()
Traceback (most recent call last):
  File "<pyshell#70>", line 1, in <module>
    spam.sort()
TypeError: unorderable types: str() < int()
```

Em terceiro lugar, sort() utiliza a "ordem ASCII" em vez da ordem alfabética para ordenar strings. Isso quer dizer que as letras maiúsculas vêm antes das letras minúsculas. Sendo assim, a letra *a* minúscula é ordenada de

modo que ela virá *após* a letra *Z* maiúscula. Para ver um exemplo, digite o seguinte no shell interativo:

```
>>> spam = ['Alice', 'ants', 'Bob', 'badgers', 'Carol', 'cats']
>>> spam.sort()
>>> spam
['Alice', 'Bob', 'Carol', 'ants', 'badgers', 'cats']
```

Se houver necessidade de ordenar os valores em ordem alfabética normal, passe str.lower para o argumento nomeado key na chamada ao método sort().

```
>>> spam = ['a', 'z', 'A', 'Z']
>>> spam.sort(key=str.lower)
>>> spam
['a', 'A', 'z', 'Z']
```

Isso fará a função sort() tratar todos os itens da lista como se tivessem letras minúsculas, sem realmente alterar os valores da lista.

Exemplo de programa: Magic 8 Ball com uma lista

Ao usar listas, podemos criar uma versão muito mais elegante do programa Magic 8 Ball do capítulo anterior. Em vez de ter diversas linhas de instruções elif quase idênticas, podemos criar uma única lista com a qual o código trabalhará. Abra uma nova janela no editor de arquivo e insira o código a seguir. Salve-o como *magic8Ball2.py*.

```
import random

messages = ['It is certain',
    'It is decidedly so',
    'Yes definitely',
    'Reply hazy try again',
    'Ask again later',
    'Concentrate and ask again',
    'My reply is no',
    'Outlook not so good',
    'Very doubtful']

print(messages[random.randint(0, len(messages) - 1)])
```

Ao executar esse programa, você verá que ele funciona do mesmo modo que o programa *magic8Ball.py* anterior.

Observe a expressão usada como índice de messages: random.randint(0, len(messages) - 1). Isso gera um número aleatório a ser usado como índice, independentemente do tamanho de messages, ou seja, você obterá um número aleatório entre 0 e o valor len(messages) - 1. A vantagem dessa abordagem é que você pode facilmente adicionar e remover strings da lista messages sem alterar outras linhas de código. Se o seu código for atualizado mais tarde, haverá menos linhas a serem alteradas e menos chances de introduzir bugs.

EXCEÇÕES ÀS REGRAS DE INDENTAÇÃO EM PYTHON

Na maioria dos casos, o nível de indentação de uma linha de código informa o Python a que bloco esse código pertence. Porém há algumas exceções a essa regra. Por exemplo, as listas podem ocupar diversas linhas no arquivo de código-fonte. A indentação dessas linhas não importa; o Python sabe que, até que o colchete de fechamento seja identificado, a lista não terá terminado. Por exemplo, você pode ter um código semelhante a:

```
spam = ['apples',
    'oranges',
                'bananas',
'cats']
print(spam)
```

É claro que, falando de modo prático, a maioria das pessoas utiliza o comportamento do Python para deixar suas listas elegantes e legíveis, como a lista de mensagens do programa Magic 8 Ball.

Também podemos dividir uma única instrução em várias linhas usando o caractere \ de continuação de linha no final. Pense na \ como se ela dissesse “esta instrução continua na próxima linha”. A indentação da linha após o caractere \ de continuação de linha não é significativa. Por exemplo, o código a seguir é válido em Python:

```
print('Four score and seven ' + \
      'years ago...')
```

Esses truques são úteis quando queremos reorganizar linhas de código Python longas a fim de torná-las um pouco mais legíveis.

Tipos semelhantes a listas: strings e tuplas

As listas não são os únicos tipos de dados que representam sequências ordenadas de valores. Por exemplo, as strings e as listas, na realidade, são semelhantes se considerarmos uma string como uma "lista" de caracteres textuais únicos. Muitas das ações que podemos realizar em listas também podem ser executadas em strings: indexação, slicing, uso em loops for com len() e uso de operadores in e not in. Para ver isso em ação, digite o seguinte no shell interativo:

```
>>> name = 'Zophie'
>>> name[0]
'Z'
>>> name[-2]
'i'
>>> name[0:4]
'Zoph'
>>> 'Zo' in name
True
>>> 'z' in name
False
>>> 'p' not in name
False
>>> for i in name:
        print('* * * ' + i + ' * * *')


* * * Z * * *
* * * o * * *
* * * p * * *
* * * h * * *
* * * i * * *
* * * e * * *
```

Tipos de dados mutáveis e imutáveis

Apesar do que foi dito, as listas e as strings são diferentes quanto a um aspecto importante. Um valor de lista é um tipo de dado *mutável*: ele pode ter valores adicionados, removidos ou alterados. No entanto uma string é

imutável: ela não pode ser alterada. Tentar atribuir novamente um único caractere em uma string resulta em um erro TypeError, como podemos ver ao digitar o seguinte no shell interativo:

```
>>> name = 'Zophie a cat'
>>> name[7] = 'the'
Traceback (most recent call last):
  File "<pyshell#50>", line 1, in <module>
    name[7] = 'the'
TypeError: 'str' object does not support item assignment
```

A maneira apropriada de efetuar uma "mutação" em uma string é usar slicing e concatenação para criar uma *nova* string, copiando partes da string antiga. Digite o seguinte no shell interativo:

```
>>> name = 'Zophie a cat'
>>> newName = name[0:7] + 'the' + name[8:12]
>>> name
'Zophie a cat'
>>> newName
'Zophie the cat'
```

Utilizamos [0:7] e [8:12] para fazer referência aos caracteres que não queremos substituir. Observe que a string 'Zophie a cat' original não foi modificada, pois as strings são imutáveis.

Embora um valor de lista *seja* mutável, a segunda linha do código a seguir não modifica a lista eggs:

```
>>> eggs = [1, 2, 3]
>>> eggs = [4, 5, 6]
>>> eggs
[4, 5, 6]
```

O valor de lista em eggs não está sendo alterado nesse caso; em vez disso, um valor de lista totalmente novo e diferente ([4, 5, 6]) está sobrescrevendo o valor antigo da lista ([1, 2, 3]). Isso está sendo representado na figura 4.2.

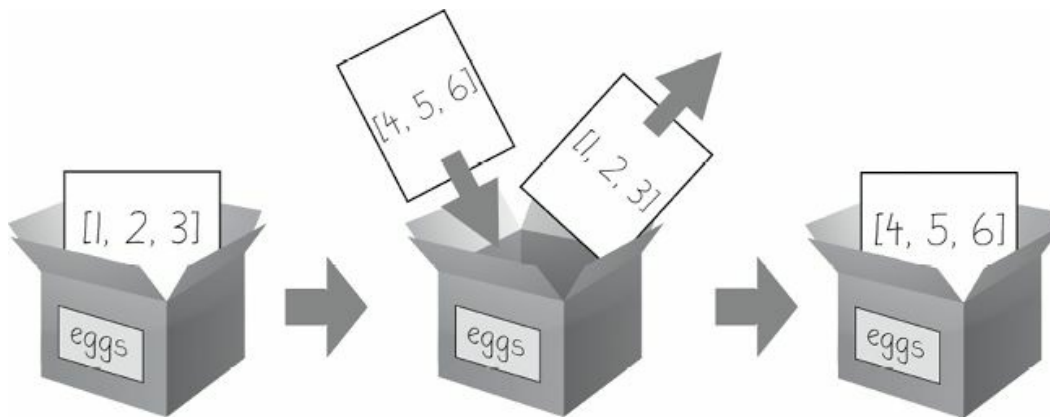



Figura 4.2 – Quando eggs = [4, 5, 6] *é executado, o conteúdo de* eggs *é substituído por um novo valor de lista.*

Se quiser realmente modificar a lista original em eggs para que ela contenha [4, 5, 6], você deverá fazer algo como:

```
>>> eggs = [1, 2, 3]
>>> del eggs[2]
>>> del eggs[1]
>>> del eggs[0]
>>> eggs.append(4)
>>> eggs.append(5)
>>> eggs.append(6)
>>> eggs
[4, 5, 6]
```

No primeiro exemplo, o valor de lista final em eggs é o mesmo valor de lista que ela tinha inicialmente. Essa lista simplesmente foi modificada, em vez de ser sobrescrita. A figura 4.3 mostra as sete alterações feitas pelas sete primeiras linhas no exemplo anterior executado no shell interativo.

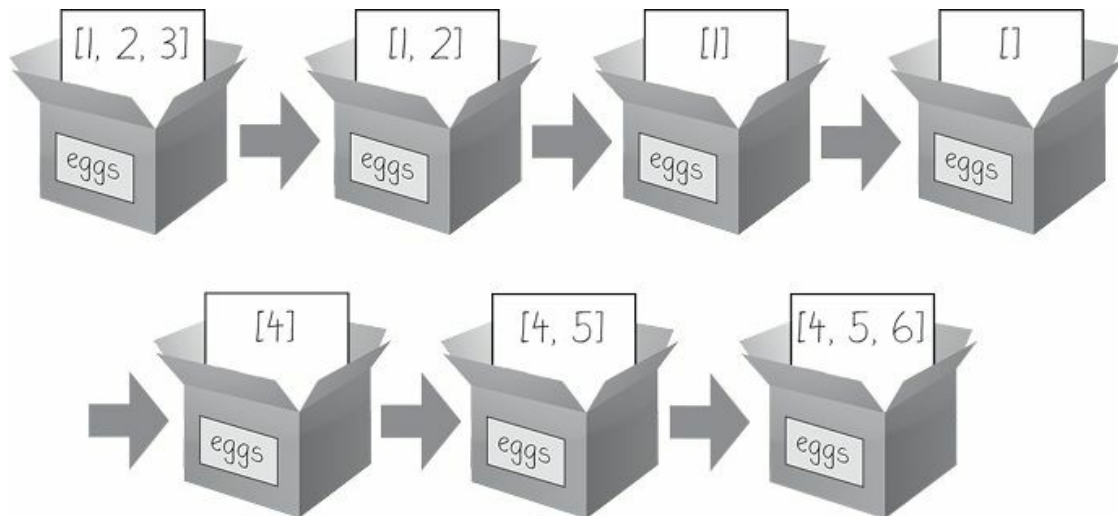



Figura 4.3 – A instrução del *e o método* append() *modificam o mesmo valor de lista in place.*

Alterar um valor de um tipo de dado mutável (como o que a instrução del e o método append() fizeram no exemplo anterior) altera o valor in place, pois o valor da variável não é substituído por um novo valor de lista.

Pode parecer que a distinção entre tipos mutáveis e os tipos imutáveis seja sem sentido, porém, na seção "Passando referências", explicaremos a diferença de comportamento existente quando chamamos funções com argumentos mutáveis em vez de chamá-las com argumentos imutáveis. Contudo, inicialmente, vamos conhecer o tipo de dado tupla (tuple), que é uma forma imutável do tipo de dado lista.

Tipo de dado tupla

O tipo de dado *tupla* (tuple) é quase idêntico ao tipo de dado lista, exceto em relação a dois aspectos. Em primeiro lugar, as tuplas são digitadas com parênteses, isto é, (e) no lugar de colchetes, ou seja, [e]. Por exemplo, digite o seguinte no shell interativo:

```
>>> eggs = ('hello', 42, 0.5)
>>> eggs[0]
'hello'
>>> eggs[1:3]
(42, 0.5)
>>> len(eggs)
3
```

Porém a principal característica que diferencia as tuplas das listas é que as tuplas, assim como as strings, são imutáveis. As tuplas não podem ter seus valores modificados, adicionados ou removidos. Digite o seguinte no shell interativo e observe a mensagem de erro TypeError:

```
>>> eggs = ('hello', 42, 0.5)
>>> eggs[1] = 99
Traceback (most recent call last):
  File "<pyshell#5>", line 1, in <module>
    eggs[1] = 99
TypeError: 'tuple' object does not support item assignment
```

Se você tiver apenas um valor em sua tupla, isso pode ser indicado por meio da inserção de uma vírgula final após o valor entre parênteses. Caso contrário, o Python achará que você simplesmente digitou um valor entre parênteses normais. A vírgula é o que informa o Python que esse valor é uma tupla. (De modo diferente de outras linguagens de programação, em Python, não há problemas em ter uma vírgula final após o último item em uma lista ou uma tupla.) Digite as seguintes chamadas à função type() no shell interativo para ver a distinção:

```
>>> type(('hello',))
<class 'tuple'>
>>> type(('hello'))
<class 'str'>
```

As tuplas podem ser usadas para informar a qualquer pessoa que estiver lendo o seu código que você não tem a intenção de mudar essa sequência de valores. Se houver necessidade de ter uma sequência ordenada de valores que não mudará nunca, utilize uma tupla. Uma segunda vantagem de usar tuplas no lugar de listas é que, pelo fato de serem imutáveis e seu conteúdo não se alterar, o Python poderá implementar algumas otimizações que deixarão os códigos que utilizam tuplas um pouco mais rápidos que os códigos que usem listas.

Convertendo tipos com as funções list() e tuple()

Assim como str(42) retorna '42', que é a representação em string do inteiro 42,

as funções list() e tuple() retornarão as versões de lista e de tupla dos valores passados a elas. Digite o seguinte no shell interativo e observe que o valor de retorno apresenta um tipo de dado diferente do valor passado:

```
>>> tuple(['cat', 'dog', 5])
('cat', 'dog', 5)
>>> list(('cat', 'dog', 5))
['cat', 'dog', 5]
>>> list('hello')
['h', 'e', 'l', 'l', 'o']
```

Converter uma tupla em uma lista será conveniente se você precisar de uma versão mutável de um valor de tupla.

Referências

Como vimos, as variáveis armazenam strings e valores inteiros. Digite o seguinte no shell interativo:

```
>>> spam = 42
>>> cheese = spam
>>> spam = 100
>>> spam
100
>>> cheese
42
```

Atribuímos o valor 42 à variável spam e, em seguida, copiamos o valor de spam e o atribuímos à variável cheese. Quando alteramos posteriormente o valor de spam para 100, isso não afeta o valor em cheese. Isso ocorre porque spam e cheese são variáveis diferentes que armazenam valores diferentes.

Porém as listas não funcionam dessa maneira. Ao atribuir uma lista a uma variável, na verdade, você estará atribuindo uma *referência* de lista à variável. Uma referência é um valor que aponta para uma porção de dados, e uma referência de lista é um valor que aponta para uma lista. A seguir, apresentamos um código que deixará essa distinção mais fácil de entender. Digite o seguinte no shell interativo:

```
u >>> spam = [0, 1, 2, 3, 4, 5]
```

```
v >>> cheese = spam
w >>> cheese[1] = 'Hello!'
  >>> spam
  [0, 'Hello!', 2, 3, 4, 5]
  >>> cheese
  [0, 'Hello!', 2, 3, 4, 5]
```

Você poderá achar isso estranho. O código alterou somente a lista cheese, porém parece que tanto a lista cheese quanto spam foram alteradas.

Quando a lista foi criada u, atribuímos uma referência a ela na variável spam. Porém a próxima linha v copia somente a referência da lista em spam para cheese, e não o valor da lista propriamente dito. Isso quer dizer que os valores armazenados em spam e em cheese agora fazem referência à mesma lista. Há apenas uma lista subjacente, pois a lista em si jamais foi realmente copiada. Sendo assim, quando modificamos o primeiro elemento de cheese w, modificamos a mesma lista referenciada por spam.

Lembre-se de que as variáveis são como caixas que contêm valores. As figuras anteriores deste capítulo mostram que listas em caixas não são exatamente representações precisas, pois as variáveis do tipo lista não contêm realmente as listas – elas contêm *referências* às listas. (Essas referências têm números de ID usados internamente pelo Python, porém você poderá ignorá-los.) Usando caixas como metáfora para as variáveis, a figura 4.4 mostra o que acontece quando uma lista é atribuída à variável spam.

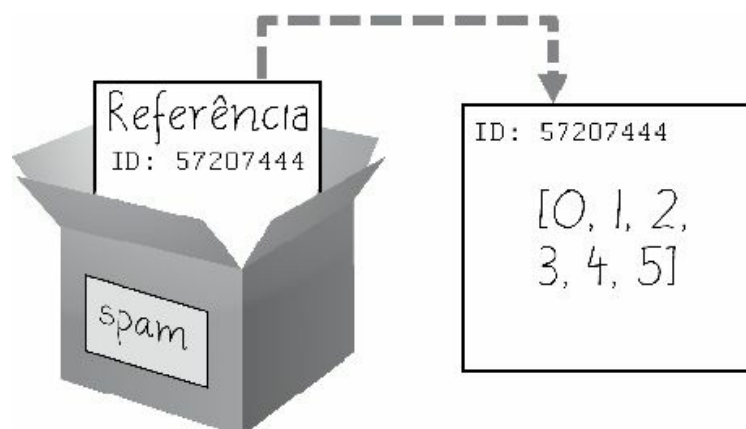



Figura 4.4 – spam = [0, 1, 2, 3, 4, 5] *armazena uma referência a uma lista, e não a lista propriamente dita.*

Então, na figura 4.5, a referência em spam é copiada para cheese. Somente

uma nova referência foi criada e armazenada em cheese, e não uma nova lista. Observe como ambas referenciam a mesma lista.

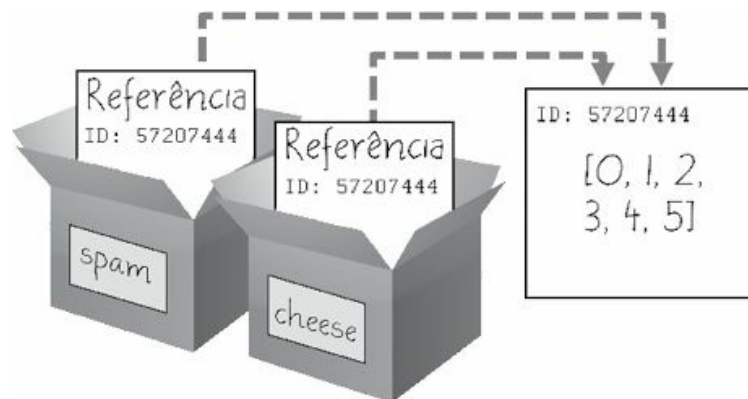



Figura 4.5 – spam = cheese *copia a referência, e não a lista.*

Ao alterar a lista referenciada por cheese, a lista a que spam se refere também é alterada, pois tanto cheese quanto spam referenciam a mesma lista. Isso pode ser visto na figura 4.6.

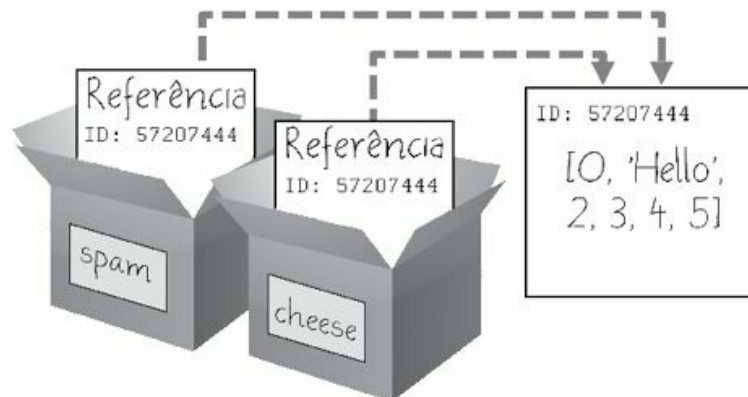



Figura 4.6 – cheese[1] = 'Hello!' *modifica a lista referenciada por ambas as variáveis.*

As variáveis conterão referências a valores de lista, e não valores de lista propriamente ditos. Porém, no caso de valores do tipo string e inteiros, as variáveis simplesmente contêm a string ou o valor inteiro. O Python utiliza referências sempre que as variáveis precisam armazenar valores de tipos de dados mutáveis, como listas ou dicionários. Para valores de tipos de dados imutáveis como strings, inteiros ou tuplas, as variáveis Python armazenarão o próprio valor.

Embora as variáveis Python tecnicamente contenham referências a valores de lista ou de dicionário, com frequência, as pessoas dizem casualmente que

a variável contém a lista ou o dicionário.

Passando referências

As referências são particularmente importantes para entender como os argumentos são passados às funções. Quando uma função é chamada, os valores dos argumentos são copiados para as variáveis referentes aos parâmetros. No caso das listas (e dos dicionários, que serão descritos no próximo capítulo), isso quer dizer que uma cópia da referência será usada para o parâmetro. Para ver as consequências disso, abra uma nova janela no editor de arquivo, digite o código a seguir e salve-o como *passingReference.py*:

```
def eggs(someParameter):
    someParameter.append('Hello')

spam = [1, 2, 3]
eggs(spam)
print(spam)
```

Observe que, quando eggs() é chamada, um valor de retorno não é usado para atribuir um novo valor a spam. Em vez disso, a lista é modificada in place, ou seja, diretamente. Quando executado, esse programa gera o seguinte resultado:

```
[1, 2, 3, 'Hello']
```

Apesar de spam e someParameter conterem referências separadas, ambos fazem referência à mesma lista. É por isso que a chamada ao método append('Hello') na função afeta a lista, mesmo após o retorno da chamada à função.

Tenha esse comportamento em mente: esquecer-se de que o Python trata variáveis de lista e de dicionário dessa maneira pode resultar em bugs confusos.

Funções copy() e deepcopy() do módulo copy

Embora passar referências por aí, em geral, seja a maneira mais prática de trabalhar com listas e dicionários, se a função modificar a lista ou o

dicionário sendo passado, talvez você não queira ter essas alterações no valor original da lista ou do dicionário. Para isso, o Python disponibiliza um módulo chamado copy que contém as funções copy() e deepcopy(). A primeira função, copy.copy(), pode ser usada para criar uma duplicata de um valor mutável como uma lista ou um dicionário, e não apenas a cópia de uma referência. Digite o seguinte no shell interativo:

```
>>> import copy
>>> spam = ['A', 'B', 'C', 'D']
>>> cheese = copy.copy(spam)
>>> cheese[1] = 42
>>> spam
['A', 'B', 'C', 'D']
>>> cheese
['A', 42, 'C', 'D']
```

Agora as variáveis spam e cheese se referem a listas diferentes, motivo pelo qual somente a lista em cheese é modificada quando atribuímos 42 ao índice 1. Como podemos ver na figura 4.7, os números de ID das referências não são mais iguais para ambas as variáveis, pois elas referenciam listas independentes.

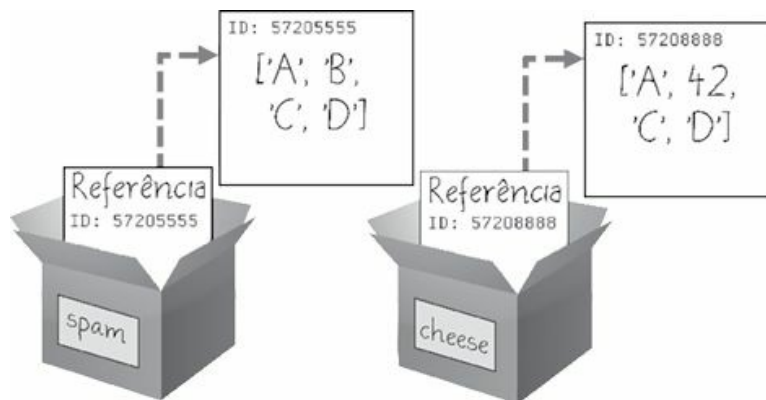



Figura 4.7 – cheese = copy.copy(spam) *cria uma segunda lista que pode ser modificada de forma independente da primeira.*

Se a lista que você precisa copiar contiver listas, utilize a função copy.deepcopy() no lugar de copy.copy(). A função deepcopy() copiará essas listas internas também.

Resumo

As listas são tipos de dados úteis, pois permitem escrever códigos que trabalhem com uma quantidade de valores que possa mudar usando uma única variável. Mais adiante neste livro, veremos programas que usam listas para realizar tarefas que seriam difíceis ou impossíveis de serem feitas sem elas.

As listas são mutáveis, o que quer dizer que seus conteúdos podem ser alterados. As tuplas e as strings, apesar de serem semelhantes às listas em alguns aspectos, são imutáveis e não podem ser alteradas. Uma variável que contenha uma tupla ou um valor do tipo string pode ser sobrescrita com uma nova tupla ou um novo valor do tipo string, porém isso não é o mesmo que modificar o valor existente in place – por exemplo, como os métodos append() ou remove() fazem nas listas.

As variáveis não armazenam valores de lista diretamente; elas armazenam *referências* às listas. Essa é uma distinção importante quando copiamos variáveis ou passamos listas como argumentos em chamadas de funções. Como o valor sendo copiado é uma referência à lista, esteja ciente de que qualquer alteração feita em uma lista poderá causar impactos em outra variável de seu programa. copy() ou deepcopy() poderão ser utilizados se você quiser fazer alterações em uma lista em uma variável sem modificar a lista original.

Exercícios práticos

1. O que é []?
2. Como você atribuiria o valor 'hello' como o terceiro valor de uma lista armazenada em uma variável chamada spam? (Suponha que spam contenha [2, 4, 6, 8, 10].)

Para as três perguntas a seguir, vamos supor que spam contenha a lista ['a', 'b', 'c', 'd'].

3. Para que valor spam[int(int('3' * 2) / 11)] é avaliado?
4. Para que valor spam[-1] é avaliado?
5. Para que valor spam[:2] é avaliado?

Para as três perguntas a seguir, vamos supor que bacon contenha a lista [3.14, 'cat', 11, 'cat', True].

6. Para que valor bacon.index('cat') é avaliado?
7. Como bacon.append(99) altera o valor de lista em bacon?
8. Como bacon.remove('cat') altera o valor de lista em bacon?
9. Quais são os operadores para concatenação de lista e para repetição de lista?
10. Qual é a diferença entre os métodos de lista append() e insert()?
11. Quais são as duas maneiras de remover valores de uma lista?
12. Nomeie alguns aspectos em relação aos quais os valores de lista são semelhantes aos valores de string.
13. Qual é a diferença entre listas e tuplas?
14. Como você deve digitar o valor de uma tupla que contenha somente o valor inteiro 42?
15. Como podemos obter a forma de tupla de um valor de lista? Como podemos obter a forma de lista de um valor de tupla?
16. As variáveis que “contêm” valores de lista não contêm realmente as listas diretamente. O que elas contêm em seu lugar?
17. Qual é a diferença entre copy.copy() e copy.deepcopy()?

Projetos práticos

Para exercitar, escreva programas que executem as tarefas a seguir.

Código para vírgulas

Suponha que você tenha um valor de lista como:

```
spam = ['apples', 'bananas', 'tofu', 'cats']
```

Crie uma função que aceite um valor de lista como argumento e retorne uma string com todos os itens separados por uma vírgula e um espaço, com *and* inserido antes do último item. Por exemplo, se passarmos a lista spam anterior à função, 'apples, bananas, tofu, and cats' será retornado. Porém sua função deverá ser capaz de trabalhar com qualquer valor de lista que ela

receber.

Grade para imagem composta de caracteres

Suponha que você tenha uma lista de listas em que cada valor das listas internas seja uma string de um caractere como:

```
grid = [['.', '.', '.', '.', '.', '.'],
        ['.', 'O', 'O', '.', '.', '.'],
        ['O', 'O', 'O', 'O', '.', '.'],
        ['O', 'O', 'O', 'O', 'O', '.'],
        ['.', 'O', 'O', 'O', 'O', 'O'],
        ['O', 'O', 'O', 'O', 'O', '.'],
        ['O', 'O', 'O', 'O', '.', '.'],
        ['.', 'O', 'O', '.', '.', '.'],
        ['.', '.', '.', '.', '.', '.']]
```

Podemos pensar em grid[x][y] como sendo o caractere nas coordenadas x e y de uma "imagem" desenhada com caracteres textuais. A origem (0, 0) estará no canto superior esquerdo, as coordenadas x aumentam para a direita e as coordenadas y aumentam para baixo.

Copie o valor da grade anterior e crie um código que a utilize para exibir a imagem:

```
..OO.OO..
.OOOOOOO.
.OOOOOOO.
..OOOOO..
...OOO...
....O....
```

Dica: você deverá usar um loop em um loop para exibir grid[0][0], em seguida grid[1][0], grid[2][0] e assim por diante, até grid[8][0]. Com isso, a primeira linha estará concluída, portanto exiba uma quebra de linha. Em seguida, seu programa deverá exibir grid[0][1], depois grid[1][1], grid[2][1] e assim por diante. O último item que seu programa exibirá é grid[8][5].

Além disso, lembre-se de passar o argumento nomeado end para print() se não quiser que uma quebra de linha seja exibida automaticamente após cada chamada a print().

CAPÍTULO 5

DICIONÁRIOS E ESTRUTURAÇÃO DE DADOS

Neste capítulo, discutirei o tipo de dado dicionário, que oferece uma maneira flexível de acessar e organizar dados. Em seguida, combinando dicionários e seu conhecimento sobre listas do capítulo anterior, aprenderemos a criar uma estrutura de dados para modelar um tabuleiro de jogo da velha.

Tipo de dado dicionário

Assim como uma lista, um *dicionário* (dictionary) é uma coleção de diversos valores. Porém, de modo diferente das listas, os índices dos dicionários podem utilizar vários tipos de dados diferentes, e não apenas inteiros. Os índices nos dicionários são chamados de *chaves* (keys), e uma chave juntamente com seu valor associado é chamada de *par chave-valor* (key-value pair)

No código, um dicionário é digitado com chaves, isto é, {}. Digite o seguinte no shell interativo:

```
>>> myCat = {'size': 'fat', 'color': 'gray', 'disposition': 'loud'}
```

Isso atribui um dicionário à variável myCat. As chaves desse dicionário são 'size', 'color' e 'disposition'. Os valores dessas chaves são 'fat', 'gray' e 'loud', respectivamente. Esses valores podem ser acessados por meio de suas chaves:

```
>>> myCat['size']
'fat'
>>> 'My cat has ' + myCat['color'] + ' fur.'
'My cat has gray fur.'
```

Os dicionários também podem usar valores inteiros como chaves, assim como as listas utilizam inteiros para os índices, porém eles não precisam começar em 0 e qualquer número pode ser usado.

```
>>> spam = {12345: 'Luggage Combination', 42: 'The Answer'}
```

Comparação entre dicionários e listas

De modo diferente das listas, os itens de um dicionários não estão ordenados. O primeiro item de uma lista chamada spam será spam[0]. Entretanto não há um “primeiro” item em um dicionário. Enquanto a ordem dos itens é importante para determinar se duas listas são iguais, não importa em que ordem os pares chave-valor são digitados em um dicionário. Digite o seguinte no shell interativo:

```
>>> spam = ['cats', 'dogs', 'moose']
>>> bacon = ['dogs', 'moose', 'cats']
>>> spam == bacon
False
>>> eggs = {'name': 'Zophie', 'species': 'cat', 'age': '8'}
>>> ham = {'species': 'cat', 'age': '8', 'name': 'Zophie'}
>>> eggs == ham
True
```

Pelo fato de não serem ordenados, os dicionários podem ser fatiados como as listas.

Tentar acessar uma chave inexistente em um dicionário resultará em uma mensagem de erro KeyError, muito semelhante à mensagem de erro IndexError referente a “fora do intervalo” em uma lista. Digite o seguinte no shell interativo e observe a mensagem de erro que aparece pelo fato de não haver nenhuma chave 'color':

```
>>> spam = {'name': 'Zophie', 'age': 7}
>>> spam['color']
Traceback (most recent call last):
  File "<pyshell#1>", line 1, in <module>
    spam['color']
KeyError: 'color'
```

Embora os dicionários não sejam ordenados, o fato de poder haver valores arbitrários para as chaves permite organizar seus dados de maneiras eficientes. Suponha que você queira que seu programa armazene dados relativos às datas de aniversário de seus amigos. Podemos usar um dicionário com os nomes como chaves e as datas de aniversário como valores. Abra uma nova janela no editor de arquivo e insira o código a seguir. Salve-o como

birthdays.py.

```
u birthdays = {'Alice': 'Apr 1', 'Bob': 'Dec 12', 'Carol': 'Mar 4'}

  while True:
      print('Enter a name: (blank to quit)')
      name = input()
      if name == '':
          break

v     if name in birthdays:
w         print(birthdays[name] + ' is the birthday of ' + name)
      else:
          print('I do not have birthday information for ' + name)
          print('What is their birthday?')
          bday = input()
x         birthdays[name] = bday
          print('Birthday database updated.')
```

Criamos um dicionário inicial e o armazenamos em birthdays u. Podemos ver se o nome fornecido existe como chave no dicionário usando a palavra-chave in v, exatamente como fizemos com as listas. Se o nome estiver no dicionário, podemos acessar o valor associado usando colchetes w; se não estiver, podemos adicioná-lo usando a mesma sintaxe de colchetes, juntamente com o operador de atribuição x.

Ao executar esse programa, a saída será semelhante a:

```
Enter a name: (blank to quit)
Alice
Apr 1 is the birthday of Alice
Enter a name: (blank to quit)
Eve
I do not have birthday information for Eve
What is their birthday?
Dec 5
Birthday database updated.
Enter a name: (blank to quit)
Eve
Dec 5 is the birthday of Eve
Enter a name: (blank to quit)
```

É claro que todos os dados fornecidos a esse programa serão esquecidos quando o programa terminar. Aprenderemos mais como salvar dados em arquivos no disco rígido no capítulo 8.

Métodos keys(), values() e items()

Há três métodos de dicionário que retornam valores semelhantes a listas contendo as chaves, os valores ou ambos – ou seja, as chaves e os valores – do dicionário: keys(), values() e items(). Os valores retornados por esses métodos não são listas de verdade: eles não podem ser modificados e não têm um método append(). Porém esses tipos de dados (dict_keys, dict_values e dict_items, respectivamente) *podem* ser usados em loops for. Para ver como esses métodos funcionam, digite o seguinte no shell interativo:

```
>>> spam = {'color': 'red', 'age': 42}
>>> for v in spam.values():
        print(v)

red
42
```

Nesse caso, um loop for faz uma iteração, percorrendo cada um dos valores do dicionário spam. Um loop for pode fazer uma iteração passando pelas chaves ou pelas chaves e pelos valores:

```
>>> for k in spam.keys():
        print(k)

color
age
>>> for i in spam.items():
        print(i)

('color', 'red')
('age', 42)
```

Usando os métodos keys(), values() e items(), um loop for pode fazer uma iteração pelas chaves, pelos valores ou pelos pares chave-valor de um dicionário, respectivamente. Observe que os valores no valor dict_items

retornado pelo método items() são tuplas contendo a chave e o valor.

Se quiser ter uma lista de verdade a partir de um desses métodos, passe o valor de retorno semelhante à lista à função list(). Digite o seguinte no shell interativo:

```
>>> spam = {'color': 'red', 'age': 42}
>>> spam.keys()
dict_keys(['color', 'age'])
>>> list(spam.keys())
['color', 'age']
```

A linha list(spam.keys()) aceita o valor dict_keys retornado por keys() e o passa para list(), que, por sua vez, retorna um valor de lista igual a ['color', 'age'].

Podemos também utilizar o truque da atribuição múltipla em um loop for para atribuir a chave e o valor a variáveis diferentes. Digite o seguinte no shell interativo:

```
>>> spam = {'color': 'red', 'age': 42}
>>> for k, v in spam.items():
        print('Key: ' + k + ' Value: ' + str(v))

Key: age Value: 42
Key: color Value: red
```

Verificando se uma chave ou um valor estão presentes em um dicionário

Lembre-se de que, de acordo com o capítulo anterior, os operadores in e not in podem verificar se um valor está presente em uma lista. Esses operadores também podem ser usados para verificar se determinada chave ou um valor estão presentes em um dicionário. Digite o seguinte no shell interativo:

```
>>> spam = {'name': 'Zophie', 'age': 7}
>>> 'name' in spam.keys()
True
>>> 'Zophie' in spam.values()
True
>>> 'color' in spam.keys()
False
```

```
>>> 'color' not in spam.keys()
True
>>> 'color' in spam
False
```

No exemplo anterior, observe que 'color' in spam é essencialmente uma versão mais concisa de 'color' in spam.keys(). É sempre isso que acontece: se quiser verificar se um valor é (ou não) uma chave do dicionário, basta usar a palavra-chave in (ou not in) com o próprio valor do dicionário.

Método get()

É tedioso verificar se uma chave está presente em um dicionário antes de acessar o valor dessa chave. Felizmente, os dicionários têm um método get() que aceita dois argumentos: a chave do valor a ser obtido e um valor alternativo a ser retornado se essa chave não existir.

Digite o seguinte no shell interativo:

```
>>> picnicItems = {'apples': 5, 'cups': 2}
>>> 'I am bringing ' + str(picnicItems.get('cups', 0)) + ' cups.'
'I am bringing 2 cups.'
>>> 'I am bringing ' + str(picnicItems.get('eggs', 0)) + ' eggs.'
'I am bringing 0 eggs.'
```

Como não há nenhuma chave 'eggs' no dicionário picnicItems, o valor default 0 é retornado pelo método get(). Sem o uso de get(), o código teria provocado uma mensagem de erro, como no exemplo a seguir:

```
>>> picnicItems = {'apples': 5, 'cups': 2}
>>> 'I am bringing ' + str(picnicItems['eggs']) + ' eggs.'
Traceback (most recent call last):
  File "<pyshell#34>", line 1, in <module>
    'I am bringing ' + str(picnicItems['eggs']) + ' eggs.'
KeyError: 'eggs'
```

Método setdefault()

Com frequência, será necessário definir um valor em um dicionário para uma chave específica somente se essa chave ainda não tiver um valor. O código será semelhante a:

```
spam = {'name': 'Pooka', 'age': 5}
if 'color' not in spam:
    spam['color'] = 'black'
```

O método setdefault() oferece uma maneira de fazer isso em uma linha de código. O primeiro argumento passado para o método é a chave a ser verificada e o segundo argumento é o valor a ser definido nessa chave caso ela não exista. Se a chave existir, o método setdefault() retornará o valor da chave. Digite o seguinte no shell interativo:

```
>>> spam = {'name': 'Pooka', 'age': 5}
>>> spam.setdefault('color', 'black')
'black'
>>> spam
{'color': 'black', 'age': 5, 'name': 'Pooka'}
>>> spam.setdefault('color', 'white')
'black'
>>> spam
{'color': 'black', 'age': 5, 'name': 'Pooka'}
```

Na primeira vez que setdefault() é chamado, o dicionário em spam altera-se para {'color': 'black', 'age': 5, 'name': 'Pooka'}. O método retorna o valor 'black', pois esse agora é o valor definido para a chave 'color'. Quando spam.setdefault('color', 'white') é chamado a seguir, o valor dessa chave *não* é alterado para 'white', pois spam já tem uma chave de nome 'color'.

O método setdefault() é um bom atalho para garantir que uma chave existe. Eis um pequeno programa que conta o número de ocorrências de cada letra em uma string. Abra a janela do editor de arquivo e insira o código a seguir, salvando-o como *characterCount.py*:

```
message = 'It was a bright cold day in April, and the clocks were striking thirteen.'
count = {}

for character in message:
    count.setdefault(character, 0)
    count[character] = count[character] + 1

print(count)
```

O programa percorre todos os caracteres da string contida na variável message em um loop, contabilizando cada caractere presente. A chamada ao método setdefault() garante que a chave está no dicionário count (com um valor default igual a 0) para que o programa não lance um erro KeyError quando count[character] = count[character] + 1 for executado. Ao executar esse programa, a saída será semelhante a:

```
{' ': 13, ',': 1, '.': 1, 'A': 1, 'I': 1, 'a': 4, 'c': 3, 'b': 1, 'e': 5, 'd': 3, 'g': 2, 'i': 6, 'h': 3, 'k': 2, 'l': 3, 'o': 2, 'n': 4, 'p': 1, 's': 3, 'r': 5, 't': 6, 'w': 2, 'y': 1}
```

De acordo com a saída, podemos ver que a letra *c* minúscula aparece três vezes, o caractere de espaço aparece treze vezes e a letra *A* maiúscula aparece uma vez. Esse programa funcionará independentemente da string presente na variável message, mesmo que a string contenha milhões de caracteres!

Apresentação elegante

Se o módulo pprint for importado em seus programas, você terá acesso às funções pprint() e pformat(), que farão uma "apresentação elegante" (pretty print) dos valores de um dicionário. Isso será conveniente quando quisermos uma apresentação mais limpa dos itens de um dicionário em comparação com o que é proporcionado por print(). Modifique o programa *characterCount.py* anterior e salve-o como *prettyCharacterCount.py*.

```
import pprint
message = 'It was a bright cold day in April, and the clocks were striking thirteen.'
count = {}

for character in message:
    count.setdefault(character, 0)
    count[character] = count[character] + 1

pprint.pprint(count)
```

Dessa vez, quando o programa for executado, a saída será muito mais limpa, com as chaves ordenadas.

```
{' ': 13,
',': 1,
```

```
'.': 1,
'A': 1,
'I': 1,
'a': 4,
'b': 1,
'c': 3,
'd': 3,
'e': 5,
'g': 2,
'h': 3,
'i': 6,
'k': 2,
'l': 3,
'n': 4,
'o': 2,
'p': 1,
'r': 5,
's': 3,
't': 6,
'w': 2,
'y': 1}
```

A função pprint.pprint() é especialmente útil quando o dicionário contém listas ou dicionários aninhados.

Se quiser obter o texto mais elegante como um valor do tipo string em vez de exibi-lo na tela, chame pprint.pformat(). As duas linhas a seguir são equivalentes:

```
pprint.pprint(someDictionaryValue)
print(pprint.pformat(someDictionaryValue))
```

Utilizando estruturas de dados para modelar objetos do mundo real

Mesmo antes da Internet, era possível jogar xadrez com alguém do outro lado do mundo. Cada jogador poderia organizar um tabuleiro de xadrez em sua casa, e eles se alternariam enviando uma carta um ao outro descrevendo cada movimento. Para isso, os jogadores precisavam de uma maneira de descrever o estado do tabuleiro e de seus movimentos sem que houvesse ambiguidade.

Na *notação algébrica do xadrez*, os espaços do tabuleiro são identificados por uma coordenada composta de um número e uma letra, como mostra a figura 5.1.

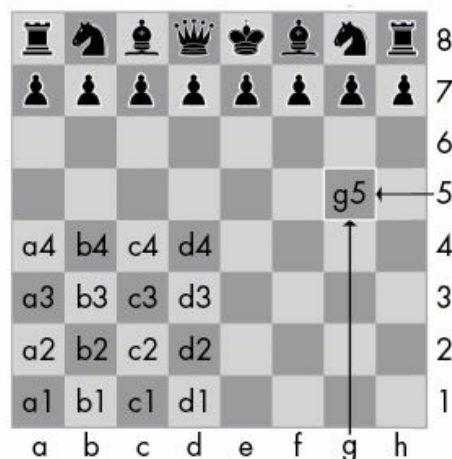



Figura 5.1 – As coordenadas de um tabuleiro na notação algébrica do xadrez.

As peças do xadrez são identificadas por letras: *K* para o rei (king), *Q* para a rainha (queen), *R* para a torre (rook), *B* para o bispo (bishop) e *N* para o cavalo (knight). Para descrever um movimento, utilizamos a letra da peça e as coordenadas de seu destino. Um par desses movimentos descreve o que acontece em uma única jogada (com as peças brancas iniciando); por exemplo, a notação *2. Nf3 Nc6* informa que o lado branco moveu um cavalo para f3 e o lado preto moveu um cavalo para c6 na segunda jogada da partida.

Há mais informações sobre a notação algébrica do que foi descrito, porém a questão é que podemos usá-la para descrever um jogo de xadrez sem que haja ambiguidades e sem a necessidade de estar diante de um tabuleiro de xadrez. Seu oponente poderia estar até mesmo do outro lado do mundo! Com efeito, não é necessário nem mesmo ter um tabuleiro físico montado se você tiver uma boa memória: basta ler os movimentos recebidos por correspondência e atualizar os tabuleiros que você tiver em sua imaginação.

Os computadores têm boa memória. Um programa em um computador moderno pode armazenar facilmente bilhões de strings como '2. Nf3 Nc6'. É assim que os computadores podem jogar xadrez sem terem um tabuleiro físico. Eles modelam os dados para que representem um tabuleiro de xadrez, e você pode escrever códigos que trabalhem com esse modelo.

É em casos como esse que as listas e os dicionários entram em cena. Eles podem ser usados para modelar objetos do mundo real, por exemplo, os tabuleiros de xadrez. Como primeiro exemplo, utilizaremos um jogo muito mais simples do que o xadrez: o jogo da velha.

Um tabuleiro de jogo da velha

Um tabuleiro de jogo da velha se parece com um símbolo grande de sustenido (#) com nove posições que podem conter um *X*, um *O* ou um espaço em branco. Para representar o tabuleiro com um dicionário, podemos atribuir uma string como chave a cada posição, conforme mostrado na figura 5.2.

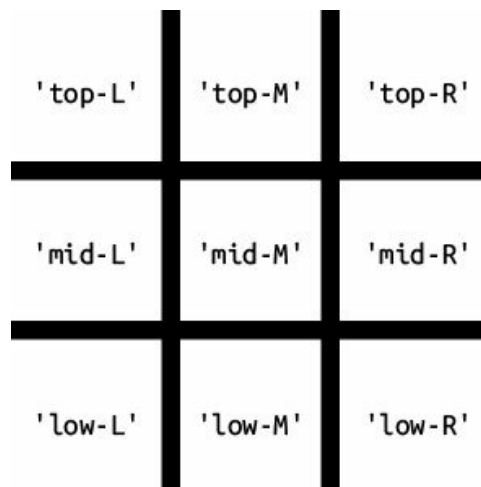



Figura 5.2 – As posições de um tabuleiro de jogo da velha, com as chaves correspondentes.

Esses valores na forma de string podem ser utilizados para representar o que cada posição contém: 'X', 'O' ou ' ' (um caractere de espaço). Desse modo, será necessário armazenar nove strings. Podemos usar um dicionário de valores para isso. O valor de string com a chave 'top-R' pode representar o canto superior direito, o valor de string com a chave 'low-L' pode representar o canto inferior esquerdo, o valor de string com a chave 'mid-M' pode representar o meio e assim por diante.

Esse dicionário contém uma estrutura de dados que representa um tabuleiro de jogo da velha. Armazene esse tabuleiro em forma de dicionário em uma variável chamada theBoard. Abra uma nova janela no editor de arquivo e insira

o código-fonte a seguir, salvando-o como *ticTacToe.py*:

```
theBoard = {'top-L': ' ', 'top-M': ' ', 'top-R': ' ',
            'mid-L': ' ', 'mid-M': ' ', 'mid-R': ' ',
            'low-L': ' ', 'low-M': ' ', 'low-R': ' '}
```

A estrutura de dados armazenada na variável theBoard representa o tabuleiro de jogo da velha da figura 5.3.

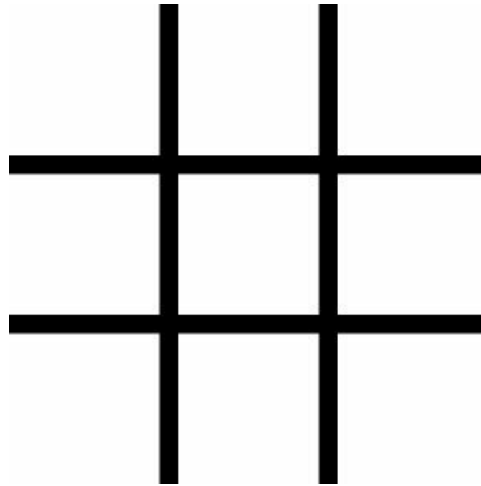



Figura 5.3 – Um tabuleiro de jogo da velha vazio.

Como o valor de todas as chaves em theBoard corresponde a uma string com um único caractere de espaço, esse dicionário representa um tabuleiro totalmente limpo. Se o jogador X inicialmente selecionar o espaço do meio, podemos representar esse tabuleiro com o dicionário a seguir:

```
theBoard = {'top-L': ' ', 'top-M': ' ', 'top-R': ' ',
            'mid-L': ' ', 'mid-M': 'X', 'mid-R': ' ',
            'low-L': ' ', 'low-M': ' ', 'low-R': ' '}
```

A estrutura de dados em theBoard agora representa o tabuleiro de jogo da velha da figura 5.4.

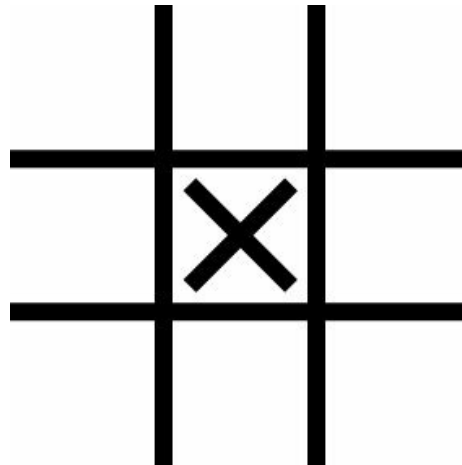



Figura 5.4 – A primeira jogada.

Um tabuleiro em que o jogador O venceu colocando *O*s na fila superior terá o seguinte aspecto:

```
theBoard = {'top-L': 'O', 'top-M': 'O', 'top-R': 'O',
         'mid-L': 'X', 'mid-M': 'X', 'mid-R': ' ',
         'low-L': ' ', 'low-M': ' ', 'low-R': 'X'}
```

A estrutura de dados em theBoard agora representa o tabuleiro de jogo da velha da figura 5.5.

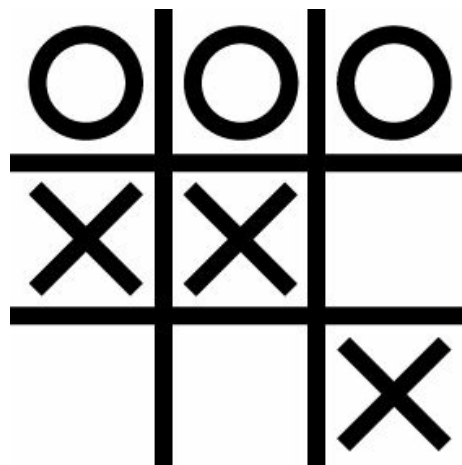



Figura 5.5 – O jogador O venceu.

É claro que o jogador vê somente o que for exibido na tela, e não o conteúdo das variáveis. Vamos criar uma função para exibir o dicionário referente ao tabuleiro na tela. Faça o seguinte acréscimo a *ticTacToe.py* (o código novo está em negrito):

```
theBoard = {'top-L': ' ', 'top-M': ' ', 'top-R': ' ',
            'mid-L': ' ', 'mid-M': ' ', 'mid-R': ' ',
            'low-L': ' ', 'low-M': ' ', 'low-R': ' '}
def printBoard(board):
    print(board['top-L'] + '|' + board['top-M'] + '|' + board['top-R'])
    print('-+-+-')
    print(board['mid-L'] + '|' + board['mid-M'] + '|' + board['mid-R'])
    print('-+-+-')
    print(board['low-L'] + '|' + board['low-M'] + '|' + board['low-R'])
printBoard(theBoard)
```

Ao executar esse programa, printBoard() exibirá um tabuleiro de jogo da velha vazio.

```
 | |
-+-+-
 | |
-+-+-
 | |
```

A função printBoard() pode tratar qualquer estrutura de dados para jogo da velha que você lhe passar. Experimente alterar o código para o seguinte:

```
theBoard = {'top-L': 'O', 'top-M': 'O', 'top-R': 'O', 'mid-L': 'X', 'mid-M': 'X', 'mid-R': ' ', 'low-L': ' ', 'low-M': ' ', 'low-R': 'X'}

def printBoard(board):
    print(board['top-L'] + '|' + board['top-M'] + '|' + board['top-R'])
    print('-+-+-')
    print(board['mid-L'] + '|' + board['mid-M'] + '|' + board['mid-R'])
    print('-+-+-')
    print(board['low-L'] + '|' + board['low-M'] + '|' + board['low-R'])
printBoard(theBoard)
```

Agora, ao executar esse programa, o novo tabuleiro será exibido na tela.

```
O|O|O
-+-+-
X|X|
-+-+-
 | |X
```

Como criamos uma estrutura de dados para representar um tabuleiro de

jogo da velha e implementamos código em printBoard() para interpretar essa estrutura de dados, você agora tem um programa que "modela" o tabuleiro de jogo da velha. Você poderia ter organizado sua estrutura de dados de modo diferente (por exemplo, usando chaves como 'TOP-LEFT' no lugar de 'top-L'), porém, desde que o código funcione com suas estruturas de dados, você terá um programa funcionando corretamente.

Por exemplo, a função printBoard() espera que a estrutura de dados para o jogo da velha seja um dicionário com chaves para todas as nove posições. Se o dicionário que você passar não contiver, por exemplo, a chave 'mid-L', seu programa não funcionará.

```
O|O|O
-+-+-
Traceback (most recent call last):
  File "ticTacToe.py", line 10, in <module>
    printBoard(theBoard)
  File "ticTacToe.py", line 6, in printBoard
    print(board['mid-L'] + '|' + board['mid-M'] + '|' + board['mid-R'])
KeyError: 'mid-L'
```

Vamos agora acrescentar o código que permita que os jogadores forneçam suas jogadas. Modifique o programa *ticTacToe.py* para que tenha o seguinte aspecto:

```
theBoard = {'top-L': ' ', 'top-M': ' ', 'top-R': ' ', 'mid-L': ' ', 'mid-M': ' ', 'mid-R': ' ',
'low-L': ' ', 'low-M': ' ', 'low-R': ' '}

def printBoard(board):
    print(board['top-L'] + '|' + board['top-M'] + '|' + board['top-R'])
    print('-+-+-')
    print(board['mid-L'] + '|' + board['mid-M'] + '|' + board['mid-R'])
    print('-+-+-')
    print(board['low-L'] + '|' + board['low-M'] + '|' + board['low-R'])
turn = 'X'
for i in range(9):
u    printBoard(theBoard)
     print('Turn for ' + turn + '. Move on which space?')
v    move = input()
w    theBoard[move] = turn
```

```
x    if turn == 'X':
         turn = 'O'
     else:
         turn = 'X'
printBoard(theBoard)
```

O novo código exibe o tabuleiro no início de cada nova jogada u, obtém o movimento do jogador ativo v, atualiza o tabuleiro de acordo com essa jogada w e, em seguida, muda o jogador ativo x antes de prosseguir para a próxima jogada.

Ao executar esse programa, a saída será semelhante a:

```
| |
-+-+-
| |
-+-+-
| |
Turn for X. Move on which space?
mid-M
| |
-+-+-
|X|
-+-+-
| |
Turn for O. Move on which space?
low-L
| |
-+-+-
|X|
-+-+-
O| |

--trecho removido--

O|O|X
-+-+-
X|X|O
-+-+-
O| |X
Turn for X. Move on which space?
```

```
low-M
O|O|X
-+-+-
X|X|O
-+-+-
O|X|X
```

Esse não é um jogo da velha completo – por exemplo, ele nem mesmo verifica se um jogador venceu –, porém é suficiente para ver como as estruturas de dados podem ser utilizadas nos programas.

NOTA Se estiver curioso, o código-fonte de um programa de jogo da velha completo está descrito nos recursos disponíveis em *http://nostarch.com/automatestuff/*.

Dicionários e listas aninhados

Modelar um tabuleiro de jogo da velha foi bem simples: o tabuleiro precisou somente de um único valor de dicionário contendo nove pares chave-valor. À medida que modelar objetos mais complexos, você perceberá que precisará de dicionários e listas que contenham outros dicionários e listas. As listas são úteis para conter uma série ordenada de valores, enquanto os dicionários são convenientes para associar chaves a valores. Por exemplo, a seguir, apresentamos um programa que utiliza um dicionário contendo outros dicionários para ver quem está trazendo o quê para um piquenique. A função totalBrought() pode ler essa estrutura de dados e calcular a quantidade total de um item trazida por todos os convidados.

```
  allGuests = {'Alice': {'apples': 5, 'pretzels': 12},
               'Bob': {'ham sandwiches': 3, 'apples': 2},
               'Carol': {'cups': 3, 'apple pies': 1}}

  def totalBrought(guests, item):
      numBrought = 0
u     for k, v in guests.items():
v         numBrought = numBrought + v.get(item, 0)
      return numBrought

  print('Number of things being brought:')
  print(' - Apples         ' + str(totalBrought(allGuests, 'apples')))
  print(' - Cups           ' + str(totalBrought(allGuests, 'cups')))
```

```
print(' - Cakes         ' + str(totalBrought(allGuests, 'cakes')))
print(' - Ham Sandwiches ' + str(totalBrought(allGuests, 'ham sandwiches')))
print(' - Apple Pies     ' + str(totalBrought(allGuests, 'apple pies')))
```

Na função totalBrought(), o loop for faz uma iteração pelos pares chave-valor em guests u. No loop, a string com o nome do convidado é atribuída a k e o dicionário de itens de piquenique que os convidados estão trazendo é atribuído a v. Se o parâmetro referente ao item estiver presente como uma chave nesse dicionário, seu valor (a quantidade) será somado a numBrought v. Se ele não existir como chave, o método get() retornará 0, que será somado a numBrought.

A saída desse programa terá a seguinte aparência:

```
Number of things being brought:
- Apples 7
- Cups 3
- Cakes 0
- Ham Sandwiches 3
- Apple Pies     1
```

Pode parecer algo simples para modelar, a ponto de você achar que não precisaria se dar o trabalho de escrever um programa para isso. Entretanto perceba que essa mesma função totalBrought() poderia facilmente tratar um dicionário que contenha milhares de convidados, cada um trazendo *milhares* de itens diferentes para o piquenique. Nesse caso, ter essas informações em uma estrutura de dados, juntamente com a função totalBrought(), fará você economizar bastante tempo!

Você pode modelar objetos com estruturas de dados de qualquer maneira que quiser, desde que o restante do código de seu programa possa trabalhar corretamente com o modelo de dados. Quando começar a programar, não se preocupe muito com a maneira "correta" de modelar os dados. À medida que adquirir mais experiência, você poderá criar modelos mais eficientes, porém o importante é que o modelo de dados funcione de acordo com as necessidades de seu programa.

Resumo

Aprendemos tudo sobre dicionários neste capítulo. As listas e os dicionários são valores que podem conter diversos valores, incluindo outras listas e outros dicionários. Os dicionários são úteis porque podemos mapear um item (a chave) a outro (o valor), diferente das listas, que simplesmente contêm uma série de valores ordenados. Os valores em um dicionário são acessados por meio de colchetes, assim como nas listas. Em vez de um índice inteiro, os dicionários podem ter chaves que sejam de uma variedade de tipos de dados: inteiros, números de ponto flutuante, strings ou tuplas. Ao organizar os valores de um programa em estruturas de dados, podemos criar representações de objetos do mundo real. Vimos um exemplo disso com um tabuleiro de jogo da velha.

Você continuará a conhecer novos conceitos no restante deste livro, porém agora já sabe o suficiente para começar a escrever alguns programas úteis que possam automatizar algumas tarefas. Talvez você não ache que tenha conhecimentos suficientes de Python para realizar tarefas como fazer download de páginas web, atualizar planilhas ou enviar mensagens de texto; contudo é nesses casos que os módulos Python entram em cena! Esses módulos, criados por outros programadores, disponibilizam funções que ajudam a realizar todas essas tarefas. Sendo assim, vamos aprender a criar programas de verdade para realizar tarefas úteis de forma automatizada.

Exercícios práticos

1. Qual é a aparência do código para criar um dicionário vazio?
2. Qual é a aparência de um valor de dicionário com uma chave igual a 'foo' e um valor 42?
3. Qual é a principal diferença entre um dicionário e uma lista?
4. O que acontecerá se você tentar acessar spam['foo'] se spam for igual a {'bar': 100}?
5. Se um dicionário estiver armazenado em spam, qual será a diferença entre as expressões 'cat' in spam e 'cat' in spam.keys()?
6. Se um dicionário estiver armazenado em spam, qual será a diferença entre as expressões 'cat' in spam e 'cat' in spam.values()?
7. Qual seria um atalho para o código a seguir?

```
if 'color' not in spam:
    spam['color'] = 'black'
```

8. Qual módulo e qual função podem ser usados para fazer uma “apresentação elegante” (pretty print) dos valores do dicionário?

Projetos práticos

Para exercitar, escreva programas que executem as tarefas a seguir.

Inventário de um jogo de fantasia

Você está criando um videogame de fantasia. A estrutura de dados para modelar o inventário do jogador será um dicionário em que as chaves são valores de string que descrevem o item do inventário e o valor será um inteiro detalhando quantos itens desse tipo o jogador tem. Por exemplo, o valor de dicionário {'rope': 1, 'torch': 6, 'gold coin': 42, 'dagger': 1, 'arrow': 12} quer dizer que o jogador tem 1 corda (rope), 6 tochas (torches), 42 moedas de ouro (gold coins) e assim por diante.

Crie uma função chamada displayInventory() que possa receber qualquer “inventário” possível e exiba essas informações da seguinte maneira:

```
Inventory:
12 arrow
42 gold coin
1 rope
6 torch
1 dagger
Total number of items: 62
```

Dica: você pode utilizar um loop for para percorrer todas as chaves de um dicionário.

```
# inventory.py
stuff = {'rope': 1, 'torch': 6, 'gold coin': 42, 'dagger': 1, 'arrow': 12}

def displayInventory(inventory):
    print("Inventory:")
    item_total = 0
```

```
    for k, v in inventory.items():
        print(str(v) + ' ' + k)
        item_total += v
    print("Total number of items: " + str(item_total))

displayInventory(stuff)
```

Função de “lista para dicionário” para o inventário de jogo de fantasia

Suponha que os despojos de um dragão vencido seja representado como uma lista de strings como esta:

```
dragonLoot = ['gold coin', 'dagger', 'gold coin', 'gold coin', 'ruby']
```

Crie uma função chamada addToInventory(inventory, addedItems), em que o parâmetro inventory seja um dicionário representando o inventário do jogador (como no projeto anterior) e o parâmetro addedItems seja uma lista como dragonLoot. A função addToInventory() deve retornar um dicionário que represente o inventário atualizado. Observe que a lista addedItems pode conter vários itens iguais. Seu código poderá ser semelhante a:

```
def addToInventory(inventory, addedItems):
    # seu código deve ser inserido aqui

inv = {'gold coin': 42, 'rope': 1}
dragonLoot = ['gold coin', 'dagger', 'gold coin', 'gold coin', 'ruby']
inv = addToInventory(inv, dragonLoot)
displayInventory(inv)
```

O programa anterior (com sua função displayInventory() do projeto anterior) apresentará a saída a seguir:

```
Inventory:
45 gold coin
1 rope
1 ruby
1 dagger

Total number of items: 48
```

CAPÍTULO 6

MANIPULAÇÃO DE STRINGS

O texto é uma das formas mais comuns de dados com as quais seus programas lidarão. Você já sabe concatenar dois valores do tipo string usando o operador +, mas poderá fazer muito mais que isso. Você poderá extrair strings parciais a partir de valores

do tipo string, adicionar ou remover espaços, fazer conversão para letras minúsculas ou maiúsculas e verificar se as strings estão formatadas corretamente. Você poderá até mesmo criar códigos Python para acessar o clipboard (área de transferência), copiar e colar textos.

Neste capítulo, aprenderemos tudo isso e muito mais. Em seguida, você trabalhará com dois projetos diferentes de programação: um gerenciador simples de senhas e um programa para automatizar a tarefa maçante que é formatar partes de um texto.

Trabalhando com strings

Vamos dar uma olhada em algumas maneiras pelas quais o Python permite escrever, exibir e acessar strings em seu código.

Strings literais

Digitar valores de string no código Python é bem simples: elas começam e terminam com aspas simples. Mas como é possível utilizar aspas simples dentro de uma string? Digitar **'That is Alice's cat.'** não funcionará, pois o Python achará que a string termina após Alice e que o restante (s cat.') é um código Python inválido. Felizmente, há diversas maneiras de digitar strings.

Aspas duplas

As strings podem começar e terminar com aspas duplas, assim como ocorre com as aspas simples. Uma vantagem de usar aspas duplas está no fato de a string poder conter um caractere de aspas simples. Digite o seguinte no shell

interativo:

```
>>> spam = "That is Alice's cat."
```

Como a string começa com aspas duplas, o Python sabe que o caractere de aspas simples faz parte da string e não marca o seu final. Entretanto, se houver necessidade de usar tanto aspas simples quanto aspas duplas na string, será preciso utilizar caracteres de escape.

Caracteres de escape

Um *caractere de escape* permite usar caracteres que, de outra maneira, não poderiam ser incluídos em uma string. Um caractere de escape é constituído de uma barra invertida (\) seguida do caractere que você deseja incluir na string. (Apesar de ser constituído de dois caracteres, é comum referenciar esse conjunto como um caractere de escape no singular.) Por exemplo, o caractere de escape para aspas simples é \'. Podemos usar isso em uma string que comece e termine com aspas simples. Para ver como os caracteres de escape funcionam, digite o seguinte no shell interativo:

```
>>> spam = 'Say hi to Bob\'s mother.'
```

O Python sabe que, como o caractere de aspas simples em Bob\'s tem uma barra invertida, ele não representa as aspas simples usadas para indicar o fim do valor da string. Os caracteres de escape \' e \" permitem inserir aspas simples e aspas duplas em suas strings, respectivamente.

A tabela 6.1 lista os caracteres de escape que podem ser usados.

Tabela 6.1 – Caracteres de escape

Caractere de escape	Exibido como
\'	Aspas simples
\"	Aspas duplas
\t	Tabulação
\n	Quebra ou mudança de linha
\\	Barra invertida

Digite o seguinte no shell interativo:

```
>>> print("Hello there!\nHow are you?\nI\'m doing fine.")
```

```
Hello there!
How are you?
I'm doing fine.
```

Strings puras

Podemos inserir um r antes das aspas de início em uma string para transformá-la em uma string pura. Uma *string pura* (raw string) ignora todos os caracteres de escape e exibe qualquer barra invertida que estiver na string. Por exemplo, digite o seguinte no shell interativo:

```
>>> print(r'That is Carol\'s cat.')
That is Carol\'s cat.
```

Pelo fato de ser uma string pura, o Python considera a barra invertida como parte da string, e não como o início de um caractere de escape. As strings puras serão úteis se você estiver digitando valores de string que contenham muitas barras invertidas, por exemplo, as strings usadas para expressões regulares descritas no próximo capítulo.

Strings de múltiplas linhas com aspas triplas

Embora seja possível usar o caractere de escape \n para inserir uma quebra de linha em uma string, geralmente, é mais fácil utilizar strings de múltiplas linhas (multiline). Uma string de múltiplas linhas em Python começa e termina com três aspas simples ou três aspas duplas. Quaisquer aspas, tabulações ou quebras de linha entre as "aspas triplas" serão consideradas parte da string. As regras de indentação do Python para blocos não se aplicam a linhas dentro de uma string de múltiplas linhas.

Abra o editor de arquivo e digite o seguinte:

```
print('''Dear Alice,

Eve's cat has been arrested for catnapping, cat burglary, and extortion.

Sincerely,
Bob''')
```

Salve esse programa como *catnapping.py* e execute-o. A saída será semelhante a:

```
Dear Alice,

Eve's cat has been arrested for catnapping, cat burglary, and extortion.

Sincerely,
Bob
```

Observe que o caractere único de aspas simples em Eve's não precisa ser escapado. Escapar aspas simples e duplas é opcional em strings multinhas. A chamada a print() a seguir exibirá um texto idêntico, porém não utiliza uma string de múltiplas linhas:

```
print('Dear Alice,\n\nEve\'s cat has been arrested for catnapping, cat burglary, and
extortion.\n\nSincerely,\nBob')
```

Comentários de múltiplas linhas

Enquanto o caractere de sustenido (#) marca o início de um comentário que inclui o restante da linha, uma string de múltiplas linhas geralmente é usada para comentários que ocupem várias linhas. O código a seguir é perfeitamente válido em Python:

```
"""Este é um programa Python para testes.
Criado por Al Sweigart al@inventwithpython.com

Esse programa foi criado para Python 3, e não para Python 2.
"""

def spam():
    """Este é um comentário de múltiplas linhas para ajudar a
    explicar o que a função spam() faz."""
    print('Hello!')
```

Indexação e slicing de strings

As strings usam índices e slices (fatias) do mesmo modo que as listas. Podemos pensar na string 'Hello world!' como uma lista e em cada caractere da string como um item com um índice correspondente.

```
' H e l l o   w o r l d ! '
  0 1 2 3 4 5 6 7 8 9 10 11
```

O espaço e o ponto de exclamação são incluídos na contagem de caracteres, portanto 'Hello world!' tem 12 caracteres de tamanho, de H no índice 0 a ! no índice 11.

Digite o seguinte no shell interativo:

```
>>> spam = 'Hello world!'
>>> spam[0]
'H'
>>> spam[4]
'o'
>>> spam[-1]
'!'
>>> spam[0:5]
'Hello'
>>> spam[:5]
'Hello'
>>> spam[6:]
'world!'
```

Se um índice for especificado, você obterá o caractere nessa posição da string. Se um intervalo for especificado de um índice a outro, o índice inicial será incluído, mas não o índice final. É por isso que, se spam for 'Hello world!', spam[0:5] será 'Hello'. A substring obtida a partir de spam[0:5] inclui tudo de spam[0] a spam[4], deixando de fora o espaço no índice 5.

Observe que o slicing de uma string não modifica a string original. Podemos capturar um slice de uma variável em uma variável diferente. Experimente digitar o seguinte no shell interativo:

```
>>> spam = 'Hello world!'
>>> fizz = spam[0:5]
>>> fizz
'Hello'
```

Ao fazer o slicing e armazenar a substring resultante em outra variável, podemos ter tanto a string completa quanto a substring à mão para termos um acesso rápido e fácil.

Operadores in e not in com strings

Os operadores in e not in podem ser usados com strings, assim como em valores de lista. Uma expressão com duas strings unidas por meio de in ou de not in será avaliada como um booleano True ou False. Digite o seguinte no shell interativo:

```
>>> 'Hello' in 'Hello World'
True
>>> 'Hello' in 'Hello'
True
>>> 'HELLO' in 'Hello World'
False
>>> '' in 'spam'
True
>>> 'cats' not in 'cats and dogs'
False
```

Essas expressões testam se a primeira string (a string exata, considerando as diferenças entre letras maiúsculas e minúsculas) pode ser encontrada na segunda string.

Métodos úteis de string

Vários métodos de string analisam strings ou criam valores transformados de strings. Esta seção descreve os métodos que utilizaremos com mais frequência.

Métodos de string upper(), lower(), isupper() e islower()

Os métodos de string upper() e lower() retornam uma nova string em que todas as letras da string original foram convertidas para letras maiúsculas ou minúsculas, respectivamente. Os caracteres que não correspondem a letras permanecem inalterados na string. Digite o seguinte no shell interativo:

```
>>> spam = 'Hello world!'
>>> spam = spam.upper()
>>> spam
'HELLO WORLD!'
>>> spam = spam.lower()
>>> spam
```

```
'hello world!'
```

Observe que esses métodos não alteram a string em si, mas retornam novos valores de string. Se quiser alterar a string original, será necessário chamar upper() ou lower() na string e, em seguida, atribuir a nova string à variável em que a original estava armazenada. É por isso que devemos usar spam = spam.upper() para alterar a string em spam no lugar de utilizar simplesmente spam.upper(). (É como se uma variável eggs contivesse o valor 10. Escrever eggs + 3 não altera o valor de eggs, porém eggs = eggs + 3 o modifica.)

Os métodos upper() e lower() serão úteis caso seja necessário fazer uma comparação sem levar em conta a diferença entre letras maiúsculas e minúsculas. As strings 'great' e 'GREat' não são iguais. No entanto, no pequeno programa a seguir, não importa se o usuário digitar Great, GREAT ou grEAT, pois a string será inicialmente convertida para letras minúsculas.

```
print('How are you?')
feeling = input()
if feeling.lower() == 'great':
    print('I feel great too.')
else:
    print('I hope the rest of your day is good.')
```

Ao executar esse programa, a pergunta será exibida, e fornecer uma variação de great, por exemplo, GREat, ainda resultará na saída I feel great too. Adicionar um código ao seu programa para tratar variações ou erros nos dados de entrada do usuário, por exemplo, um uso inconsistente de letras maiúsculas, fará seus programas serem mais simples de usar e os deixará menos suscetíveis a falhas.

```
How are you?
GREat
I feel great too.
```

Os métodos isupper() e islower() retornarão um valor booleano True se a string tiver pelo menos uma letra e todas as letras forem maiúsculas ou minúsculas, respectivamente. Caso contrário, o método retornará False. Digite o seguinte no shell interativo e observe o que cada chamada de método retorna:

```
>>> spam = 'Hello world!'
```

```
>>> spam.islower()
False
>>> spam.isupper()
False
>>> 'HELLO'.isupper()
True
>>> 'abc12345'.islower()
True
>>> '12345'.islower()
False
>>> '12345'.isupper()
False
```

Como os métodos de string upper() e lower() retornam strings, também podemos chamar os métodos de string *nesses* valores de string retornados. As expressões que fazem isso se parecerão com uma cadeia de chamadas de métodos. Digite o seguinte no shell interativo:

```
>>> 'Hello'.upper()
'HELLO'
>>> 'Hello'.upper().lower()
'hello'
>>> 'Hello'.upper().lower().upper()
'HELLO'
>>> 'HELLO'.lower()
'hello'
>>> 'HELLO'.lower().islower()
True
```

Métodos de string isX

Juntamente com islower() e isupper(), há diversos métodos de string cujos nomes começam com a palavra *is*. Esses métodos retornam um valor booleano que descreve a natureza da string. Eis alguns métodos de string is*X* comuns:

- isalpha() retornará True se a string for constituída somente de letras e não estiver vazia.
- isalnum() retornará True se a string for constituída somente de letras e números e não estiver vazia.

- isdecimal() retornará True se a string for constituída somente de caracteres numéricos e não estiver vazia.
- isspace() retornará True se a string for constituída somente de espaços, tabulações e quebras de linha e não estiver vazia.
- istitle() retornará True se a string for constituída somente de palavras que comecem com uma letra maiúscula seguida somente de letras minúsculas.

Digite o seguinte no shell interativo:

```
>>> 'hello'.isalpha()
True
>>> 'hello123'.isalpha()
False
>>> 'hello123'.isalnum()
True
>>> 'hello'.isalnum()
True
>>> '123'.isdecimal()
True
>>> '    '.isspace()
True
>>> 'This Is Title Case'.istitle()
True
>>> 'This Is Title Case 123'.istitle()
True
>>> 'This Is not Title Case'.istitle()
False
>>> 'This Is NOT Title Case Either'.istitle()
False
```

Os métodos de string is*X* são úteis para validar dados de entrada do usuário. Por exemplo, o programa a seguir pergunta repetidamente aos usuários a idade e pede uma senha até que dados de entrada válidos sejam fornecidos. Abra uma nova janela no editor de arquivo e insira o programa a seguir, salvando-o como *validateInput.py*:

```
while True:
    print('Enter your age:')
    age = input()
    if age.isdecimal():
```

```
        break
    print('Please enter a number for your age.')

while True:
    print('Select a new password (letters and numbers only):')
    password = input()
    if password.isalnum():
        break
    print('Passwords can only have letters and numbers.')
```

No primeiro loop while, perguntamos a idade ao usuário e armazenamos sua entrada em age. Se age for um valor válido (decimal), sairemos desse primeiro loop while e prosseguiremos para o segundo, que pede uma senha. Caso contrário, informamos o usuário que ele deve fornecer um número e perguntamos sua idade novamente. No segundo loop while, devemos pedir uma senha, armazenar o dado de entrada do usuário em password e sair do loop se a entrada for alfanumérica. Se não for, não ficaremos satisfeitos; sendo assim, dizemos ao usuário que a senha deve ser alfanumérica e, novamente, pedimos que uma senha seja fornecida.

Ao ser executado, a saída desse programa será semelhante a:

```
Enter your age:
forty two
Please enter a number for your age.
Enter your age:
42
Select a new password (letters and numbers only):
secr3t!
Passwords can only have letters and numbers.
Select a new password (letters and numbers only):
secr3t
```

Se chamarmos isdecimal() e isalnum() em variáveis, poderemos testar se os valores armazenados nessas variáveis são decimais ou não, ou se são alfanuméricos ou não. Nesse caso, esses testes nos ajudam a rejeitar a entrada forty two e a aceitar 42, além de rejeitar secr3t! e aceitar secr3t.

Métodos de string startswith() e endswith()

Os métodos startswith() e endswith() retornarão True se o valor de string com o qual forem chamados começar ou terminar (respectivamente) com a string passada para o método; do contrário, retornarão False. Digite o seguinte no shell interativo:

```
>>> 'Hello world!'.startswith('Hello')
True
>>> 'Hello world!'.endswith('world!')
True
>>> 'abc123'.startswith('abcdef')
False
>>> 'abc123'.endswith('12')
False
>>> 'Hello world!'.startswith('Hello world!')
True
>>> 'Hello world!'.endswith('Hello world!')
True
```

Esses métodos serão alternativas convenientes ao operador == de igualdade caso seja preciso verificar se apenas a primeira ou a última parte da string, e não a string completa, é igual a outra string.

Métodos de string join() e split()

O método join() é útil quando temos uma lista de strings que devem ser unidas em um único valor de string. O método join() é chamado em uma string, recebe uma lista de strings e retorna uma string. A string retornada corresponde à concatenação de todas as strings da lista passada para o método. Por exemplo, digite o seguinte no shell interativo:

```
>>> ', '.join(['cats', 'rats', 'bats'])
'cats, rats, bats'
>>> ' '.join(['My', 'name', 'is', 'Simon'])
'My name is Simon'
>>> 'ABC'.join(['My', 'name', 'is', 'Simon'])
'MyABCnameABCisABCSimon'
```

Observe que a string em que join() é chamada é inserida entre cada string do argumento de lista. Por exemplo, quando join(['cats', 'rats', 'bats']) é chamada na string ', ', a string retornada é 'cats, rats, bats'.

Lembre-se de que join() é chamado em um valor de string e recebe um valor de lista. (É fácil chamar acidentalmente de modo invertido.) O método split() faz o inverso: é chamado em um valor de string e retorna uma lista de strings. Digite o seguinte no shell interativo:

```
>>> 'My name is Simon'.split()
['My', 'name', 'is', 'Simon']
```

Por padrão, a string 'My name is Simon' é separada sempre que caracteres em branco, como caracteres de espaço, tabulação ou quebra de linha, forem encontrados. Esses caracteres de espaço em branco não são incluídos nas strings da lista retornada. Podemos passar uma string delimitadora ao método split() para especificar uma string diferente em relação à qual a separação será feita. Por exemplo, digite o seguinte no shell interativo:

```
>>> 'MyABCnameABCisABCSimon'.split('ABC')
['My', 'name', 'is', 'Simon']
>>> 'My name is Simon'.split('m')
['My na', 'e is Si', 'on']
```

Um uso comum de split() está em dividir uma string de múltiplas linhas nos caracteres de quebra de linha. Digite o seguinte no shell interativo:

```
>>> spam = '''Dear Alice,
How have you been? I am fine.
There is a container in the fridge
that is labeled "Milk Experiment".
Please do not drink it.
Sincerely,
Bob'''
>>> spam.split('\n')
['Dear Alice,', 'How have you been? I am fine.', 'There is a container in the
fridge', 'that is labeled "Milk Experiment".', '', 'Please do not drink it.',
'Sincerely,', 'Bob']
```

Passar o argumento '\n' a split() permite separar a string com múltiplas linhas armazenada em spam nas quebras de linha e retornar uma lista em que cada item corresponda a uma linha da string.

Justificando texto com rjust(), ljust() e center()

Os métodos de string rjust() e ljust() retornam uma versão preenchida da string em que são chamados, com espaços inseridos para justificar o texto. O primeiro argumento de ambos os métodos é um inteiro referente ao tamanho da string justificada. Digite o seguinte no shell interativo:

```
>>> 'Hello'.rjust(10)
'     Hello'
>>> 'Hello'.rjust(20)
'               Hello'
>>> 'Hello World'.rjust(20)
'         Hello World'
>>> 'Hello'.ljust(10)
'Hello     '
```

'Hello'.rjust(10) diz que queremos justificar 'Hello' à direita em uma string de tamanho total igual a 10. 'Hello' tem cinco caracteres, portanto cinco espaços serão acrescentados à sua esquerda, resultando em uma string de dez caracteres, com 'Hello' justificado à direita.

Um segundo argumento opcional de rjust() e ljust() especifica um caractere de preenchimento que não seja um caractere de espaço. Digite o seguinte no shell interativo:

```
>>> 'Hello'.rjust(20, '*')
'***************Hello'
>>> 'Hello'.ljust(20, '-')
'Hello---------------'
```

O método de string center() funciona como ljust() e rjust(), porém centraliza o texto em vez de justificá-lo à esquerda ou à direita. Digite o seguinte no shell interativo:

```
>>> 'Hello'.center(20)
'       Hello        '
>>> 'Hello'.center(20, '=')
'=======Hello========'
```

Esses métodos serão especialmente úteis quando for necessário exibir dados tabulares que tiverem o espaçamento correto. Abra uma nova janela no editor de arquivo e insira o código a seguir, salvando-o como *picnicTable.py*:

```
def printPicnic(itemsDict, leftWidth, rightWidth):
    print('PICNIC ITEMS'.center(leftWidth + rightWidth, '-'))
    for k, v in itemsDict.items():
        print(k.ljust(leftWidth, '.') + str(v).rjust(rightWidth))

picnicItems = {'sandwiches': 4, 'apples': 12, 'cups': 4, 'cookies': 8000}
printPicnic(picnicItems, 12, 5)
printPicnic(picnicItems, 20, 6)
```

Nesse programa, definimos um método printPicnic() que recebe um dicionário contendo informações e usa center(), ljust() e rjust() para exibir essas informações em um formato organizado e alinhado de tabela.

O dicionário que passaremos a printPicnic() é picnicItems. Em picnicItems, temos 4 sanduíches (sandwiches), 12 maçãs (apples), 4 copos (cups) e 8.000 biscoitos (cookies). Queremos organizar essas informações em duas colunas, com o nome do item à esquerda e a quantidade à direita.

Para isso, devemos decidir qual será a largura que queremos que as colunas à esquerda e à direita tenham. Juntamente com o nosso dicionário, passaremos esses valores a printPicnic().

printPicnic() recebe um dicionário, um leftWidth para a coluna esquerda de uma tabela e um rightWidth para a coluna direita. A função exibe um título PICNIC ITEMS centralizado na parte superior da tabela. Em seguida, um loop percorre o dicionário exibindo cada par chave-valor em uma linha, com a chave justificada à esquerda e preenchida com pontos e o valor justificado à direita, preenchido com espaços.

Após definir printPicnic(), criamos o dicionário picnicItems e chamamos printPicnic() duas vezes, passando larguras diferentes para as colunas da esquerda e da direita da tabela.

Ao executar esse programa, os itens do piquenique serão exibidos duas vezes. Na primeira vez, a coluna da esquerda terá 12 caracteres de largura, e a coluna da direita, 5 caracteres de largura. Na segunda vez, as colunas terão 20 e 6 caracteres de largura, respectivamente.

```
---PICNIC ITEMS--
sandwiches..    4
apples......   12
```

```
cups........    4
cookies..... 8000
-------PICNIC ITEMS-------
sandwiches..........    4
apples.............   12
cups...............    4
cookies............  8000
```

Usar rjust(), ljust() e center() permite garantir que as strings estejam elegantemente alinhadas, mesmo que você não saiba ao certo quantos caracteres têm suas strings.

Removendo espaços em branco com strip(), rstrip() e lstrip()

Às vezes, você pode querer remover caracteres de espaços em branco (espaço, tabulação e quebra de linha) do lado esquerdo, do lado direito ou de ambos os lados de uma string. O método de string strip() retornará uma nova string sem caracteres de espaços em branco no início ou no fim. Os métodos lstrip() e rstrip() removerão caracteres de espaços em branco das extremidades esquerda e direita, respectivamente. Digite o seguinte no shell interativo:

```
>>> spam = '    Hello World    '
>>> spam.strip()
'Hello World'
>>> spam.lstrip()
'Hello World    '
>>> spam.rstrip()
'    Hello World'
```

Opcionalmente, um argumento do tipo string especificará quais caracteres deverão ser removidos das extremidades. Digite o seguinte no shell interativo:

```
>>> spam = 'SpamSpamBaconSpamEggsSpamSpam'
>>> spam.strip('ampS')
'BaconSpamEggs'
```

Passar o argumento 'ampS' a strip() lhe dirá para remover as ocorrências de a, m, p e da letra S maiúscula das extremidades da string armazenada em spam. A

ordem dos caracteres na string passada para strip() não importa: strip('ampS') fará o mesmo que strip('mapS') ou strip('Spam').

Copiando e colando strings com o módulo pyperclip

O módulo pyperclip tem funções copy() e paste() capazes de enviar e receber texto do clipboard (área de transferência) de seu computador. Enviar a saída de seu programa para o clipboard facilitará colá-lo em um email, um processador de texto ou em outro software.

O pyperclip não vem com o Python. Para instalá-lo, siga as instruções para instalação de módulos de terceiros no apêndice A. Após instalar o módulo pyperclip, digite o seguinte no shell interativo:

```
>>> import pyperclip
>>> pyperclip.copy('Hello world!')
>>> pyperclip.paste()
'Hello world!'
```

É claro que, se algo fora de seu programa alterar o conteúdo do clipboard, a função paste() retornará essa informação. Por exemplo, se eu copiar esta frase para o clipboard e, em seguida, chamar paste(), o resultado terá a seguinte aparência:

```
>>> pyperclip.paste()
'Por exemplo, se eu copiar esta frase para o clipboard e, em seguida, chamar paste(), o
resultado terá a seguinte aparência:'
```

EXECUTANDO SCRIPTS PYTHON FORA DO IDLE

Até agora, executamos os scripts Python usando o shell interativo e o editor de arquivo no IDLE. Entretanto você não vai querer ter o inconveniente de abrir o IDLE e o script Python sempre que quiser executar um script. Felizmente, há atalhos que podemos configurar para facilitar a execução dos scripts Python. Os passos são um pouco diferentes para Windows, OS X e Linux, porém cada um deles está descrito no apêndice B. Consulte o apêndice B para saber como executar seus scripts Python de forma conveniente e poder passar argumentos de linha de comando a eles. (Você não poderá passar argumentos de linha de comando a seus programas usando o IDLE.)

Projeto: Repositório de senhas

É provável que você tenha contas em vários sites diferentes. Usar a mesma senha em todos eles é um péssimo hábito porque, se um desses sites tiver uma falha de segurança, os hackers saberão a senha de todas as suas demais contas. É melhor usar um software gerenciador de senhas em seu computador que utilize uma senha principal para desbloquear esse gerenciador de senhas. Então você poderá copiar qualquer senha de conta para o clipboard e colá-la no campo de senha do site.

O programa gerenciador de senhas que criaremos nesse exemplo não é seguro, porém oferece uma demonstração básica de como esses programas funcionam.

OS PROJETOS DOS CAPÍTULOS

Esse é o primeiro "projeto do capítulo" deste livro. A partir de agora, cada capítulo terá projetos que demonstrarão conceitos discutidos nesse capítulo. Os projetos são criados de modo a levarem você de uma janela em branco do editor de arquivo até um programa completo e funcional. Assim como nos exemplos com o shell interativo, não leia simplesmente as seções de projeto – execute-os em seu computador!

Passo 1: Design do programa e estruturas de dados

Queremos executar esse programa com um argumento de linha de comando correspondente ao nome da conta – por exemplo, *email* ou *blog*. A senha dessa conta será copiada para o clipboard para que o usuário possa colá-la em um campo de Senha. Dessa maneira, o usuário poderá ter senhas longas e complexas sem a necessidade de memorizá-las.

Abra uma nova janela no editor de arquivo e salve o programa como *pw.py*. O programa deve começar com uma linha contendo #! (*shebang*) – veja o apêndice B –, e você também deve escrever um comentário que descreva brevemente o programa. Como queremos associar o nome de cada conta à sua senha, podemos armazenar essas informações como strings em um dicionário. O dicionário será a estrutura de dados que organizará os dados de suas contas e senhas. Faça seu programa ter o seguinte aspecto:

```
#! python3
# pw.py – Um programa para repositório de senhas que não é seguro.

PASSWORDS = {'email': 'F7minlBDDuvMJuxESSKHFhTxFtjVB6',
          'blog': 'VmALvQyKAxiVH5G8v01if1MLZF3sdt',
          'luggage': '12345'}
```

Passo 2: Tratar argumentos da linha de comando

Os argumentos da linha de comando serão armazenados na variável sys.argv. (Veja o apêndice B para obter mais informações sobre como usar argumentos de linha de comando em seus programas.) O primeiro item da lista sys.argv sempre será uma string contendo o nome do arquivo do programa ('pw.py'), e o segundo item deverá ser o primeiro argumento da linha de comando. Nesse programa, esse argumento será o nome da conta cuja senha você deseja obter. Como o argumento de linha de comando é obrigatório, você deve exibir uma mensagem de uso ao usuário caso ele se esqueça de adicioná-lo (ou seja, se a lista sys.argv contiver menos de dois valores). Faça seu programa ter o seguinte aspecto:

```
#! python3
# pw.py – Um programa para repositório de senha que não é seguro.

PASSWORDS = {'email': 'F7minlBDDuvMJuxESSKHFhTxFtjVB6',
          'blog': 'VmALvQyKAxiVH5G8v01if1MLZF3sdt',
          'luggage': '12345'}

import sys
if len(sys.argv) < 2:
   print('Usage: python pw.py [account] - copy account password')
   sys.exit()

account = sys.argv[1]    # o primeiro argumento da linha de comando é o nome da conta
```

Passo 3: Copiar a senha correta

Agora que o nome da conta está armazenado como uma string na variável account, devemos verificar se ele existe no dicionário PASSWORDS como uma chave. Em caso afirmativo, devemos copiar o valor da chave para o clipboard

usando pyperclip.copy(). (Como o módulo pyperclip está sendo usado, é necessário importá-lo.) Observe que não *precisamos* realmente da variável account; poderíamos simplesmente usar sys.argv[1] em todos os lugares em que account é usado nesse programa. Porém uma variável chamada account é muito mais legível do que algo enigmático como sys.argv[1].

Faça seu programa ter o seguinte aspecto:

```
#! python3
# pw.py – Um programa para repositório de senhas que não é seguro.

PASSWORDS = {'email': 'F7minlBDDuvMJuxESSKHFhTxFtjVB6',
             'blog': 'VmALvQyKAxiVH5G8v01if1MLZF3sdt',
             'luggage': '12345'}

import sys, pyperclip
if len(sys.argv) < 2:
    print('Usage: py pw.py [account] - copy account password')
    sys.exit()

account = sys.argv[1]    # o primeiro argumento da linha de comando é o nome da conta

if account in PASSWORDS:
    pyperclip.copy(PASSWORDS[account])
    print('Password for ' + account + ' copied to clipboard.')
else:
    print('There is no account named ' + account)
```

Esse novo código procura o nome da conta no dicionário PASSWORDS. Se o nome da conta for uma chave no dicionário, leremos o valor correspondente a essa chave, copiaremos esse valor para o clipboard e exibiremos uma mensagem informando que o valor foi copiado. Caso contrário, exibiremos uma mensagem informando que não há nenhuma conta com esse nome.

Esse é o script completo. Ao usar as instruções do apêndice B para iniciar programas com linha de comando facilmente, você terá uma maneira rápida de copiar as senhas de suas contas para o clipboard. Será necessário modificar o valor do dicionário PASSWORDS no código-fonte sempre que você quiser atualizar o programa com uma nova senha.

É claro que, provavelmente, você não vai querer manter todas as suas

senhas em um local em que qualquer pessoa poderia copiá-las facilmente. No entanto esse programa poderá ser modificado e usado para copiar textos normais rapidamente para o clipboard. Suponha que você esteja enviando diversos emails que têm muitos dos mesmos parágrafos em comum. Você poderia colocar cada parágrafo como um valor no dicionário PASSWORDS (é provável que você queira renomear o dicionário a essa altura); desse modo, você terá uma maneira de selecionar e copiar rapidamente uma das várias partes de texto padrão para o clipboard.

No Windows, um arquivo batch poderá ser criado para executar esse programa com a janela Run de WIN-R. (Para saber mais sobre arquivos batch, consulte o apêndice B.) Digite o seguinte no editor de arquivo e salve como *pw.bat* na pasta *C:\Windows*:

```
@py.exe C:\Python34\pw.py %*
@pause
```

Com esse arquivo batch criado, executar o programa de senhas protegidas no Windows é somente uma questão de pressionar WIN-R e digitar pw <*nome da conta*>.

Projeto: Adicionando marcadores na marcação da Wiki

Ao editar um artigo na Wikipedia, podemos criar uma lista com marcações (bullets) ao inserir cada item da lista em sua própria linha e inserindo um asterisco na frente. Porém suponha que você tenha uma lista realmente extensa em que você queira acrescentar marcadores. Você poderia simplesmente digitar esses asteriscos no início de cada linha, uma a uma, ou poderia automatizar essa tarefa com um pequeno script Python.

O script *bulletPointAdder.py* obterá o texto do clipboard, adicionará um asterisco e um espaço no início de cada linha e, em seguida, colará esse novo texto no clipboard. Por exemplo, se eu copiar o texto a seguir [do artigo "List of Lists of Lists" (Listas de listas de listas) da Wikipedia] para o clipboard:

```
Lists of animals
Lists of aquarium life
Lists of biologists by author abbreviation
Lists of cultivars
```

e depois executar o programa *bulletPointAdder.py*, o clipboard conterá o seguinte:

```
* Lists of animals
* Lists of aquarium life
* Lists of biologists by author abbreviation
* Lists of cultivars
```

Esse texto contendo um asterisco como prefixo está pronto para ser colado em um artigo da Wikipedia como uma lista com marcadores.

Passo 1: Copiar e colar no clipboard

Queremos que o programa *bulletPointAdder.py* faça o seguinte:

1. Obtenha o texto do clipboard.
2. Faça algo com ele.
3. Copie o novo texto para o clipboard.

O segundo passo é um pouco complicado, porém os passos 1 e 3 são bem simples: eles envolvem somente as funções pyperclip.copy() e pyperclip.paste(). Por enquanto, vamos apenas criar a parte do programa que trata os passos 1 e 3. Digite o seguinte, salvando o programa como *bulletPointAdder.py*:

```
#! python3
# bulletPointAdder.py – Acrescenta marcadores da Wikipedia no início
# de cada linha de texto do clipboard.

import pyperclip
text = pyperclip.paste()

# TODO: Separa as linhas e acrescenta asteriscos.
pyperclip.copy(text)
```

O comentário TODO é um lembrete de que você deve completar essa parte do programa em algum momento. O próximo passo consiste em realmente implementar essa parte do programa.

Passo 2: Separar as linhas de texto e acrescentar o asterisco

A chamada a pyperclip.paste() retorna todo o texto que está no clipboard na forma de uma string extensa. Se usarmos o exemplo de “List of Lists of Lists”, a string armazenada em text terá o seguinte aspecto:

```
'Lists of animals\nLists of aquarium life\nLists of biologists by author
abbreviation\nLists of cultivars'
```

Os caracteres \n de quebra de linha nessa string fazem com que ela seja apresentada em várias linhas quando for exibida ou copiada do clipboard. Há várias “linhas” nesse único valor de string. Você deve acrescentar um asterisco no início de cada uma dessas linhas.

Poderíamos criar um código que procurasse todos os caracteres \n de quebra de linha na string e, em seguida, acrescentasse o asterisco imediatamente depois deles. Todavia será mais fácil usar o método split() para retornar uma lista de strings, uma para cada linha da string original, e então acrescentar o asterisco na frente de cada string da lista.

Faça seu programa ter o seguinte aspecto:

```
#! python3
# bulletPointAdder.py – Acrescenta marcadores da Wikipedia no início
# de cada linha de texto do clipboard.

import pyperclip
text = pyperclip.paste()

# Separa as linhas e acrescenta os asteriscos.
lines = text.split('\n')
for i in range(len(lines)):    # percorre todos os índices da lista "lines" em um loop
    lines[i] = '* ' + lines[i] # acrescenta um asterisco em cada string da lista "lines"

pyperclip.copy(text)
```

Separamos o texto nas quebras de linha para obter uma lista em que cada item corresponda a uma linha de texto. Armazenamos a lista em lines e percorremos seus itens usando um loop. Para cada linha, acrescentamos um asterisco e um espaço no início dessa linha. Agora cada string em lines começa com um asterisco.

Passo 3: Juntar as linhas modificadas

A lista lines agora contém as linhas modificadas iniciadas com asteriscos. Porém pyperclip.copy() está esperando um único valor de string, e não uma lista de valores de string. Para compor esse valor único de string, passe lines ao método join() a fim de obter uma única string resultante da junção das strings da lista. Faça seu programa ter o seguinte aspecto:

```
#! python3
# bulletPointAdder.py – Acrescenta marcadores da Wikipedia no início
# de cada linha de texto do clipboard.

import pyperclip
text = pyperclip.paste()

# Separa as linhas e acrescenta os asteriscos.
lines = text.split('\n')
for i in range(len(lines)):    # percorre todos os índices da lista "lines" em um loop
    lines[i] = '* ' + lines[i] # acrescenta um asterisco em cada string da lista "lines"
text = '\n'.join(lines)
pyperclip.copy(text)
```

Quando é executado, esse programa substitui o texto do clipboard por um texto que contém asteriscos no início de cada linha. Agora o programa está completo e você pode tentar executá-lo com um texto copiado no clipboard.

Mesmo que não precise automatizar essa tarefa específica, talvez você queira automatizar outro tipo de manipulação de textos, por exemplo, remover espaços dos finais das linhas ou converter textos para letras maiúsculas ou minúsculas. Independentemente do que você precisar, o clipboard poderá ser usado para entrada e saída de dados.

Resumo

O texto é uma forma comum de dados e o Python oferece diversos métodos úteis de string para processar um texto armazenado em valores de string. Você utilizará indexação, slicing e métodos de string em quase todos os programas Python que criar.

Os programas que você está criando agora não parecem ser muito

sofisticados – eles não têm interfaces gráficas de usuário com imagens e textos coloridos. Até agora, exibimos texto com print() e permitimos que o usuário fornecesse texto com input(). No entanto o usuário pode fornecer grandes quantidades de texto rapidamente por meio do clipboard. Essa capacidade oferece uma opção conveniente para escrever programas que manipulem grandes volumes de texto. Esses programas baseados em texto podem não ter janelas nem imagens sofisticadas, porém podem realizar diversas tarefas convenientes rapidamente.

Outra maneira de manipular grandes quantidades de texto consiste em ler e escrever em arquivos diretamente no disco rígido. Aprenderemos a fazer isso com o Python no próximo capítulo.

Exercícios práticos

1. O que são caracteres de escape?
2. O que os caracteres de escape \n e \t representam?
3. Como podemos inserir um caractere \ de barra invertida em uma string?
4. O valor de string "Howl's Moving Castle" é uma string válida. Por que não há problema no fato de o caractere único de aspas simples na palavra Howl's não estar escapado?
5. Se não quiser colocar \n em sua string, como você poderá escrever uma string contendo quebras de linha?
6. Para que valores as expressões a seguir são avaliadas?
   - 'Hello world!'[1]
   - 'Hello world!'[0:5]
   - 'Hello world!'[:5]
   - 'Hello world!'[3:]
7. Para que valores as expressões a seguir são avaliadas?
   - 'Hello'.upper()
   - 'Hello'.upper().isupper()
   - 'Hello'.upper().lower()
8. Para que valores as expressões a seguir são avaliadas?

- 'Remember, remember, the fifth of November.'.split()
- '-'.join('There can be only one.'.split())

9. Quais métodos de string podem ser usados para justificar uma string à direita, à esquerda e para centralizá-la?
10. Como podemos remover caracteres de espaços em branco no início e no fim de uma string?

Projeto prático

Para exercitar, escreva um programa que execute a seguinte tarefa.

Exibição de tabela

Crie uma função chamada printTable() que receba uma lista de listas de strings e a exiba em uma tabela bem organizada, com cada coluna justificada à direita. Suponha que todas as listas internas contenham o mesmo número de strings. Por exemplo, o valor poderá ter o seguinte aspecto:

```
tableData = [['apples', 'oranges', 'cherries', 'banana'],
             ['Alice', 'Bob', 'Carol', 'David'],
             ['dogs', 'cats', 'moose', 'goose']]
```

Sua função printTable() exibirá o seguinte:

```
  apples Alice  dogs
 oranges   Bob  cats
cherries Carol moose
  banana David goose
```

Dica: seu código inicialmente deverá localizar a string mais longa em cada uma das listas internas para que a coluna toda tenha largura suficiente para que todas as strings possam ser inseridas. Você pode armazenar a largura máxima de cada coluna como uma lista de inteiros. A função printTable() pode começar com colWidths = [0] * len(tableData), que criará uma lista contendo o mesmo número de valores 0 que o número de listas internas em tableData. Dessa maneira, colWidths[0] poderá armazenar a largura da string mais longa de tableData[0], colWidths[1] poderá armazenar a largura da string mais longa de tableData[1] e assim por diante. Você poderá então identificar o maior valor na

lista colWidths e descobrir a largura na forma de um inteiro a ser passada para o método de string rjust().

PARTE II
AUTOMATIZANDO TAREFAS

CAPÍTULO 7

CORRESPONDÊNCIA DE PADRÕES COM EXPRESSÕES REGULARES

Talvez você já esteja acostumado a pesquisar um texto pressionando CTRL-F e digitando as palavras que estiver procurando. As *expressões regulares* vão um passo além: elas permitem especificar um *padrão* de texto a ser procurado. Talvez você não saiba exatamente o número de um telefone comercial, porém, se morar nos Estados Unidos ou no Canadá, saberá que ele contém três dígitos seguidos de um hífen e depois mais quatro dígitos

(e, opcionalmente, um código de área de três dígitos no início). É assim que você como ser humano reconhece um número de telefone quando o vê: 415-555-1234 é um número de telefone, porém 4.155.551.234 não é.

As expressões regulares são úteis, mas muitos que não são programadores as desconhecem, apesar de os editores e processadores de texto mais modernos como o Microsoft Word ou o OpenOffice terem recursos de pesquisa e de pesquisa e substituição que possam fazer buscas baseadas em expressões regulares. As expressões regulares permitem economizar bastante tempo não só para os usuários de software, mas também para os programadores. Com efeito, o autor de obras técnicas Cory Doctorow argumenta que, mesmo antes de ensinar programação, devíamos ensinar expressões regulares:

> *Conhecer [as expressões regulares] pode significar a diferença entre resolver um problema em três passos e resolvê-lo em 3 mil passos. Quando se é um nerd, você esquece que os problemas que resolvemos com o pressionamento de algumas teclas podem exigir dias de trabalho lento, maçante e suscetível a erros de outras pessoas.*[1] [2]

Neste capítulo, começaremos criando um programa para encontrar padrões de texto *sem* usar expressões regulares e então veremos como usar essas expressões para deixar o código muito mais compacto. Mostrarei como fazer

correspondências básicas usando expressões regulares e, em seguida, prosseguirei apresentando alguns recursos mais eficazes como substituição de strings e criação de suas próprias classes de caracteres. Por fim, no final do capítulo, criaremos um programa que poderá extrair automaticamente números de telefone e endereços de email de um bloco de texto.

Encontrando padrões de texto sem usar expressões regulares

Suponha que você queira encontrar um número de telefone em uma string. Você conhece o padrão: três números, um hífen, três números, um hífen e quatro números. Aqui está um exemplo: 415-555-4242.

Vamos usar uma função chamada isPhoneNumber() para verificar se uma string corresponde a esse padrão, retornando True ou False. Abra uma nova janela no editor de arquivo e insira o código a seguir; salve o arquivo como *isPhoneNumber.py*:

```
  def isPhoneNumber(text):
u     if len(text) != 12:
          return False
      for i in range(0, 3):
v         if not text[i].isdecimal():
              return False
w     if text[3] != '-':
          return False
      for i in range(4, 7):
x         if not text[i].isdecimal():
              return False
y     if text[7] != '-':
          return False
      for i in range(8, 12):
z         if not text[i].isdecimal():
              return False
{     return True

  print('415-555-4242 is a phone number:')
  print(isPhoneNumber('415-555-4242'))
  print('Moshi moshi is a phone number:')
```

```
print(isPhoneNumber('Moshi moshi'))
```

Quando esse programa for executado, a saída terá o seguinte aspecto:

```
415-555-4242 is a phone number:
True
Moshi moshi is a phone number:
False
```

A função isPhoneNumber() contém um código que realiza diversas verificações para saber se a string em text é um número de telefone válido. Se alguma dessas verificações falhar, a função retornará False. Inicialmente, o código verifica se a string tem exatamente 12 caracteres u. Em seguida, verifica se o código de área (ou seja, os três primeiros caracteres em text) é constituído somente de caracteres numéricos v. O restante da função verifica se a string está de acordo com o padrão de um número de telefone: o número deve ter um primeiro hífen após o código de área w, deve ter mais três caracteres numéricos x, depois outro hífen y e, por fim, mais quatro números z. Se a execução do programa conseguir passar por todas essas verificações, True será retornado {.

Chamar isPhoneNumber() com o argumento '415-555-4242' retornará True. Chamar isPhoneNumber() com 'Moshi moshi' retornará False; o primeiro teste falha, pois 'Moshi moshi' não tem um tamanho igual a 12 caracteres.

Será necessário adicionar mais código ainda para encontrar esse padrão de texto em uma string maior. Substitua as quatro últimas chamadas à função print() em *isPhoneNumber.py* por:

```
  message = 'Call me at 415-555-1011 tomorrow. 415-555-9999 is my office.'
  for i in range(len(message)):
u     chunk = message[i:i+12]
v     if isPhoneNumber(chunk):
          print('Phone number found: ' + chunk)
  print('Done')
```

Quando esse programa for executado, a saída terá o seguinte aspecto:

```
Phone number found: 415-555-1011
Phone number found: 415-555-9999
Done
```

A cada iteração do loop for, uma nova porção de 12 caracteres de message é atribuída à variável chunk u. Por exemplo, na primeira iteração, i será 0 e chunk receberá message[0:12] (ou seja, a string 'Call me at 4'). Na próxima iteração, i será 1 e chunk receberá message[1:13] (a string 'all me at 41').

Passamos chunk a isPhoneNumber() para verificar se ela corresponde ao padrão de número de telefone v e, em caso afirmativo, essa porção da string será exibida.

Continue a percorrer message em um loop e, em algum momento, os 12 caracteres em chunk corresponderão a um número de telefone. O loop percorrerá a string toda, testando cada porção de 12 caracteres e exibindo qualquer chunk que satisfaça isPhoneNumber(). Após termos acabado de percorrer message, exibimos Done.

Embora a string em message seja curta nesse exemplo, ela poderia ter milhões de caracteres de tamanho e o programa continuaria executando em menos de um segundo. Um programa semelhante que encontra números de telefone usando expressões regulares também seria executado em menos de um segundo, porém as expressões regulares agilizam a escrita desses programas.

Encontrando padrões de texto com expressões regulares

O programa anterior para encontrar números de telefone funciona, porém utiliza bastante código para fazer algo limitado: a função isPhoneNumber() contém 17 linhas, porém é capaz de identificar somente um padrão de número de telefone. O que aconteceria se um número de telefone estivesse formatado como 415.555.4242 ou como (415) 555-4242? E se o número de telefone tivesse uma extensão, por exemplo, 415-555-4242 x99? A função isPhoneNumber() falharia ao validar esses números. Poderíamos acrescentar mais código para esses padrões adicionais, porém há uma maneira mais simples de resolver esse problema.

As expressões regulares (regular expressions), chamadas de *regexes* por questões de concisão, correspondem a descrições para um padrão de texto. Por exemplo, um \d é uma regex que representa um dígito – ou seja, qualquer numeral único entre 0 e 9. A regex \d\d\d-\d\d\d-\d\d\d\d é usada pelo Python

para fazer a correspondência do mesmo texto conforme feito pela função isPhoneNumber(): uma string com três números, um hífen, mais três números, outro hífen e quatro números. Qualquer outra string não corresponderá à regex \d\d\d-\d\d\d-\d\d\d\d.

No entanto as expressões regulares podem ser muito mais sofisticadas. Por exemplo, acrescentar um 3 entre chaves ({3}) após um padrão é como dizer "faça a correspondência desse padrão três vezes". Desse modo, a regex \d{3}-\d{3}-\d{4}, um pouco mais concisa, também corresponde ao formato correto de número de telefone.

Criando objetos Regex

Todas as funções de regex em Python estão no módulo re. Digite o seguinte no shell interativo para importar esse módulo:

```
>>> import re
```

NOTA A maioria dos próximos exemplos neste capítulo exigirá o módulo re, portanto lembre-se de importá-lo no início de qualquer script que você criar ou sempre que reiniciar o IDLE. Caso contrário, uma mensagem de erro NameError: name 're' is not defined (NameError: nome 're' não está definido) será obtida.

Passar um valor de string que represente sua expressão regular a re.compile() fará um objeto Regex de padrão ser retornado (ou, simplesmente, um objeto Regex).

Para criar um objeto Regex que corresponda ao padrão de número de telefone, insira o seguinte no shell interativo. (Lembre-se de que \d quer dizer "um caractere correspondente a um dígito" e \d\d\d-\d\d\d-\d\d\d\d é a expressão regular para o padrão correto de número de telefone.)

```
>>> phoneNumRegex = re.compile(r'\d\d\d-\d\d\d-\d\d\d\d')
```

Agora a variável phoneNumRegex contém um objeto Regex.

PASSANDO STRINGS PURAS PARA RE.COMPILE( )

Lembre-se de que os caracteres de escape em Python usam a barra invertida (\). O valor de string '\n' representa um único caractere de quebra de linha, e não uma barra invertida seguida de um *n* minúsculo. É preciso fornecer o caractere de escape \\ para exibir uma única barra invertida. Portanto '\\n' é a string que representa uma barra invertida seguida de um *n* minúsculo. Entretanto, ao colocar um r antes do primeiro caractere de aspas do valor da string, podemos marcar a string como *pura* (raw string), que não considera caracteres de escape.

Como as expressões regulares geralmente utilizam barras invertidas, é conveniente passar strings puras para a função re.compile() em vez de digitar barras invertidas extras. Digitar r'\d\d\d-\d\d\d-\d\d\d\d' é muito mais fácil do que digitar '\\d\\d\\d-\\d\\d\\d-\\d\\d\\d\\d'.

Objetos Regex de correspondência

O método search() de um objeto Regex pesquisa a string recebida em busca de qualquer correspondência com a regex. O método search() retornará None se o padrão da regex não for encontrado na string. Se o padrão *for* encontrado, o método search() retornará um objeto Match. Objetos Match têm um método group() que retornará o texto correspondente extraído da string pesquisada. (Explicarei os grupos em breve.) Por exemplo, digite o seguinte no shell interativo:

```
>>> phoneNumRegex = re.compile(r'\d\d\d-\d\d\d-\d\d\d\d')
>>> mo = phoneNumRegex.search('My number is 415-555-4242.')
>>> print('Phone number found: ' + mo.group())
Phone number found: 415-555-4242
```

O nome da variável mo é somente um nome genérico usado em objetos Match. Esse exemplo pode parecer complicado à primeira vista, porém é muito mais conciso que o programa *isPhoneNumber.py* anterior e faz o mesmo.

Nesse caso, passamos o padrão desejado a re.compile() e armazenamos o objeto Regex resultante em phoneNumRegex. Em seguida, chamamos search() em phoneNumRegex e lhe passamos a string em que queremos encontrar uma correspondência. O resultado da pesquisa será armazenado na variável mo.

Nesse exemplo, sabemos que nosso padrão será encontrado na string, portanto sabemos que um objeto Match será retornado. Sabendo que mo contém um objeto Match e não o valor nulo None, podemos chamar group() em mo para retornar a correspondência. Escrever mo.group() em nossa instrução print exibirá a correspondência completa, ou seja, 415-555-4242.

Revisão da correspondência com expressão regular

Embora haja diversos passos para usar expressões regulares em Python, cada passo é bem simples.

1. Importe o módulo de regex usando import re.
2. Crie um objeto Regex usando a função re.compile(). (Lembre-se de usar uma string pura.)
3. Passe a string que você quer pesquisar ao método search() do objeto Regex. Isso fará um objeto Match ser retornado.
4. Chame o método group() do objeto Match para retornar uma string com o texto correspondente.

> NOTA Apesar de incentivá-lo a digitar o código de exemplo no shell interativo, você também deve usar ferramentas de teste de expressões regulares baseadas em web, que poderão mostrar exatamente como uma regex faz a correspondência de uma porção de texto especificada. Recomendo usar a ferramenta de teste em *http://regexpal.com/*.

Mais correspondência de padrões com expressões regulares

Agora que os passos básicos para criar e encontrar objetos correspondentes a expressões regulares em Python já são conhecidos, você está pronto para experimentar alguns dos recursos mais eficazes da correspondência de padrões.

Agrupando com parênteses

Suponha que você queira separar o código de área do restante do número de telefone. A adição de parênteses criará *grupos* na regex: (\d\d\d)-(\d\d\d-\d\d\d\d). Então você poderá usar o método group() do objeto de

correspondência para obter o texto correspondente de apenas um grupo.

O primeiro conjunto de parênteses em uma string de regex será o grupo 1. O segundo conjunto será o grupo 2. Ao passar o inteiro 1 ou 2 ao método group() do objeto de correspondência, podemos obter partes diferentes do texto correspondente. Ao passar 0 ou nada ao método group(), o texto correspondente completo será retornado. Digite o seguinte no shell interativo:

```
>>> phoneNumRegex = re.compile(r'(\d\d\d)-(\d\d\d-\d\d\d\d)')
>>> mo = phoneNumRegex.search('My number is 415-555-4242.')
>>> mo.group(1)
'415'
>>> mo.group(2)
'555-4242'
>>> mo.group(0)
'415-555-4242'
>>> mo.group()
'415-555-4242'
```

Se quiser obter todos os grupos de uma só vez, utilize o método groups() – observe a forma do plural no nome.

```
>>> mo.groups()
('415', '555-4242')
>>> areaCode, mainNumber = mo.groups()
>>> print(areaCode)
415
>>> print(mainNumber)
555-4242
```

Como mo.groups() retorna uma tupla com diversos valores, podemos usar o truque da atribuição múltipla para atribuir cada valor a uma variável diferente, como na linha areaCode, mainNumber = mo.groups() anterior.

Os parênteses têm um significado especial em expressões regulares, porém o que devemos fazer se for necessário corresponder a parênteses em seu texto? Por exemplo, talvez os números de telefone aos quais você esteja tentando corresponder tenham o código de área definido entre parênteses. Nesse caso, será necessário escapar os caracteres (e) com uma barra invertida. Digite o seguinte no shell interativo:

```
>>> phoneNumRegex = re.compile(r'(\(\d\d\d\)) (\d\d\d-\d\d\d\d)')
>>> mo = phoneNumRegex.search('My phone number is (415) 555-4242.')
>>> mo.group(1)
'(415)'
>>> mo.group(2)
'555-4242'
```

Os caracteres de escape \(e \) na string pura passada para re.compile() corresponderão aos caracteres de parênteses propriamente ditos.

Fazendo a correspondência de vários grupos com pipe

O caractere | é chamado de *pipe*. Podemos usá-lo em qualquer lugar em que quisermos fazer a correspondência de uma entre várias expressões. Por exemplo, a expressão regular r'Batman|Tina Fey' corresponde a 'Batman' ou a 'Tina Fey'.

Quando *tanto* Batman *quanto* Tina Fey ocorrerem na string pesquisada, a primeira ocorrência do texto correspondente será retornada como o objeto Match. Digite o seguinte no shell interativo:

```
>>> heroRegex = re.compile (r'Batman|Tina Fey')
>>> mo1 = heroRegex.search('Batman and Tina Fey.')
>>> mo1.group()
'Batman'

>>> mo2 = heroRegex.search('Tina Fey and Batman.')
>>> mo2.group()
'Tina Fey'
```

NOTA Podemos encontrar *todas* as ocorrências correspondentes com o método findall() que será discutido na seção "Método findall()".

O pipe também pode ser usado para fazer a correspondência de um entre diversos padrões como parte de sua regex. Por exemplo, suponha que você queira fazer a correspondência de qualquer uma das strings 'Batman', 'Batmobile', 'Batcopter' e 'Batbat'. Como todas essas strings começam com Bat, seria interessante se você pudesse especificar esse prefixo somente uma vez. Isso pode ser feito com parênteses. Digite o seguinte no shell interativo:

```
>>> batRegex = re.compile(r'Bat(man|mobile|copter|bat)')
```

```
>>> mo = batRegex.search('Batmobile lost a wheel')
>>> mo.group()
'Batmobile'
>>> mo.group(1)
'mobile'
```

A chamada ao método mo.group() retorna o texto correspondente 'Batmobile' completo, enquanto mo.group(1) retorna somente a parte do texto correspondente dentro do primeiro grupo de parênteses, ou seja, 'mobile'. Ao usar o caractere pipe e os parênteses de agrupamento, podemos especificar diversos padrões alternativos aos quais você gostaria que sua regex correspondesse.

Se houver necessidade de fazer a correspondência de um caractere de pipe propriamente dito, escape-o com uma barra invertida, como em \|.

Correspondência opcional usando ponto de interrogação

Às vezes, há um padrão ao qual você quer corresponder somente de forma opcional. Isso quer dizer que a regex deve encontrar uma correspondência independentemente de essa porção de texto estar ou não presente. O caractere ? marca o grupo que o antecede como sendo uma parte opcional do padrão. Por exemplo, digite o seguinte no shell interativo:

```
>>> batRegex = re.compile(r'Bat(wo)?man')
>>> mo1 = batRegex.search('The Adventures of Batman')
>>> mo1.group()
'Batman'

>>> mo2 = batRegex.search('The Adventures of Batwoman')
>>> mo2.group()
'Batwoman'
```

A parte da expressão regular que contém (wo)? significa que o padrão wo é um grupo opcional. A regex corresponderá a textos que não tenham nenhuma ou que tenham uma instância de wo. É por isso que a regex corresponde tanto a 'Batwoman' quanto a 'Batman'.

Usando o exemplo anterior com o número de telefone, podemos fazer a regex procurar números de telefone que tenham ou não um código de área.

Digite o seguinte no shell interativo:

```
>>> phoneRegex = re.compile(r'(\d\d\d-)?\d\d\d-\d\d\d\d')
>>> mo1 = phoneRegex.search('My number is 415-555-4242')
>>> mo1.group()
'415-555-4242'

>>> mo2 = phoneRegex.search('My number is 555-4242')
>>> mo2.group()
'555-4242'
```

Você pode pensar no ? como se dissesse “faça a correspondência de zero ou de uma ocorrência do grupo que antecede esse ponto de interrogação”.

Se houver necessidade de fazer a correspondência de um caractere de ponto de interrogação propriamente dito, escape-o com \?.

Correspondendo a zero ou mais ocorrências usando asterisco

O * (chamado de *asterisco*) quer dizer “corresponda a zero ou mais” – o grupo que antecede o asterisco pode ocorrer qualquer número de vezes no texto. Esse grupo poderá estar totalmente ausente ou ser repetido diversas vezes. Vamos dar uma olhada no exemplo contendo Batman novamente.

```
>>> batRegex = re.compile(r'Bat(wo)*man')
>>> mo1 = batRegex.search('The Adventures of Batman')
>>> mo1.group()
'Batman'

>>> mo2 = batRegex.search('The Adventures of Batwoman')
>>> mo2.group()
'Batwoman'

>>> mo3 = batRegex.search('The Adventures of Batwowowowoman')
>>> mo3.group()
'Batwowowowoman'
```

Para 'Batman', a parte referente a (wo)* da regex corresponde a zero instâncias de wo na string; para 'Batwoman', (wo)* corresponde a uma instância de wo; e para 'Batwowowowoman', (wo)* corresponde a quatro instâncias de wo.

Se houver necessidade de fazer a correspondência do caractere asterisco propriamente dito, utilize uma barra invertida antes do asterisco na expressão regular, ou seja, *.

Correspondendo a uma ou mais ocorrências usando o sinal de adição

Enquanto * quer dizer "corresponda a zero ou mais", o + (ou *sinal de adição*) quer dizer "corresponda a um ou mais". De modo diferente do asterisco, que não exige que seu grupo esteja presente na string correspondente, o grupo que antecede um sinal de adição deve aparecer *pelo menos uma vez*. Ele não é opcional. Digite o seguinte no shell interativo e compare o resultado com as regexes que usam asterisco da seção anterior:

```
>>> batRegex = re.compile(r'Bat(wo)+man')
>>> mo1 = batRegex.search('The Adventures of Batwoman')
>>> mo1.group()
'Batwoman'

>>> mo2 = batRegex.search('The Adventures of Batwowowowoman')
>>> mo2.group()
'Batwowowowoman'

>>> mo3 = batRegex.search('The Adventures of Batman')
>>> mo3 == None
True
```

A regex Bat(wo)+man não identificará uma correspondência na string 'The Adventures of Batman', pois pelo menos um wo é exigido pelo sinal de adição.

Se houver necessidade de fazer a correspondência de um sinal de adição propriamente dito, insira uma barra invertida antes desse caractere para escapá-lo, ou seja, use \+.

Correspondendo a repetições específicas usando chaves

Se você tiver um grupo que deseja repetir um número específico de vezes, insira um número entre chaves após o grupo em sua regex. Por exemplo, a regex (Ha){3} corresponde à string 'HaHaHa', mas não a 'HaHa', pois essa última

tem apenas duas repetições do grupo (Ha).

Em vez de um número, podemos especificar um intervalo especificando um mínimo, uma vírgula e um máximo entre chaves. Por exemplo, a regex (Ha){3,5} corresponde a 'HaHaHa', 'HaHaHaHa' e 'HaHaHaHaHa'.

Também podemos deixar de fora o primeiro ou o segundo número nas chaves para deixar de especificar o mínimo ou o máximo. Por exemplo, (Ha){3,} corresponderá a três ou mais instâncias do grupo (Ha), enquanto (Ha){,5} corresponderá a zero até cinco instâncias. As chaves podem ajudar a deixar suas expressões regulares mais concisas. As duas expressões regulares a seguir correspondem a padrões idênticos:

```
(Ha){3}
(Ha)(Ha)(Ha)
```

As duas expressões regulares a seguir também correspondem a padrões idênticos:

```
(Ha){3,5}
((Ha)(Ha)(Ha))|((Ha)(Ha)(Ha)(Ha))|((Ha)(Ha)(Ha)(Ha)(Ha))
```

Digite o seguinte no shell interativo:

```
>>> haRegex = re.compile(r'(Ha){3}')
>>> mo1 = haRegex.search('HaHaHa')
>>> mo1.group()
'HaHaHa'

>>> mo2 = haRegex.search('Ha')
>>> mo2 == None
True
```

Nesse caso, (Ha){3} corresponde a 'HaHaHa', mas não a 'Ha'. Como não há correspondência em 'Ha', search() retorna None.

Correspondências greedy e nongreedy

Como (Ha){3,5} pode corresponder a três, quatro ou cinco instâncias de Ha na string 'HaHaHaHaHa', você pode estar se perguntando por que a chamada a group() do objeto Match no exemplo anterior com chaves retorna 'HaHaHaHaHa',

e não as possibilidades mais curtas. Afinal de contas, 'HaHaHa' e 'HaHaHaHa' também são correspondências válidas para a expressão regular (Ha){3,5}.

As expressões regulares em Python são *greedy* (gulosas) por default, o que significa que, em situações ambíguas, a correspondência será feita com a maior string possível. Na versão *nongreedy* (não gulosa) das chaves, que faz a correspondência com a menor string possível, um ponto de interrogação é usado depois da chave de fechamento.

Digite o seguinte no shell interativo e observe a diferença entre as formas greedy e nongreedy das chaves em que a mesma string é pesquisada:

```
>>> greedyHaRegex = re.compile(r'(Ha){3,5}')
>>> mo1 = greedyHaRegex.search('HaHaHaHaHa')
>>> mo1.group()
'HaHaHaHaHa'

>>> nongreedyHaRegex = re.compile(r'(Ha){3,5}?')
>>> mo2 = nongreedyHaRegex.search('HaHaHaHaHa')
>>> mo2.group()
'HaHaHa'
```

Observe que o ponto de interrogação pode ter dois significados em expressões regulares: declarar uma correspondência nongreedy ou indicar um grupo opcional. Esses significados não têm nenhuma relação entre si.

Método findall()

Além do método search(), os objetos Regex também têm um método findall(). Enquanto search() retorna um objeto Match do *primeiro* texto correspondente na string pesquisada, o método findall() retorna as strings de *todas* as correspondências na string pesquisada. Para ver como search() retorna um objeto Match somente da primeira instância do texto correspondente, digite o seguinte no shell interativo:

```
>>> phoneNumRegex = re.compile(r'\d\d\d-\d\d\d-\d\d\d\d')
>>> mo = phoneNumRegex.search('Cell: 415-555-9999 Work: 212-555-0000')
>>> mo.group()
'415-555-9999'
```

Por outro lado, findall() não retorna um objeto Match, mas uma lista de strings – *desde que não haja grupos na expressão regular*. Cada string da lista é uma parte do texto pesquisado que correspondeu à expressão regular. Digite o seguinte no shell interativo:

```
>>> phoneNumRegex = re.compile(r'\d\d\d-\d\d\d-\d\d\d\d') # não tem nenhum grupo
>>> phoneNumRegex.findall('Cell: 415-555-9999 Work: 212-555-0000')
['415-555-9999', '212-555-0000']
```

Se *houver* grupos na expressão regular, findall() retornará uma lista de tuplas. Cada tupla representa uma correspondência identificada, e seus itens serão as strings correspondentes a cada grupo da regex. Para ver findall() em ação, digite o seguinte no shell interativo (observe que a expressão regular sendo compilada agora contém grupos entre parênteses):

```
>>> phoneNumRegex = re.compile(r'(\d\d\d)-(\d\d\d)-(\d\d\d\d)') # tem grupos
>>> phoneNumRegex.findall('Cell: 415-555-9999 Work: 212-555-0000')
[('415', '555', '9999'), ('212', '555', '0000')]
```

Para resumir o que o método findall() retorna, lembre-se do seguinte:

1. Quando chamado em uma regex sem grupos, por exemplo, \d\d\d-\d\d\d-\d\d\d\d, o método findall() retorna uma lista de strings correspondentes, como ['415-555-9999', '212-555-0000'].
2. Quando chamado em uma regex que tenha grupos, por exemplo, (\d\d\d)-(\d\d\d)-(\d\d\d\d), o método findall() retorna uma lista de tuplas contendo strings (uma string para cada grupo), como em [('415', '555', '1122'), ('212', '555', '0000')].

Classes de caracteres

No exemplo anterior de regex para número de telefone, aprendemos que \d pode representar qualquer dígito, ou seja, \d é uma versão abreviada da expressão regular (0|1|2|3|4|5|6|7|8|9). Há várias dessas *classes abreviadas de caracteres*, conforme mostrado na tabela 7.1.

Tabela 7.1 – Códigos abreviados para classes comuns de caracteres

Classe de caracteres	Representa

abreviada	
\d	Qualquer dígito de 0 a 9.
\D	Qualquer caractere que *não* seja um dígito de 0 a 9.
\w	Qualquer letra, dígito ou o caractere underscore. (Pense nisso como uma correspondência de caracteres de “palavra”.)
\W	Qualquer caractere que *não* seja uma letra, um dígito ou o caractere underscore.
\s	Qualquer espaço, tabulação ou caractere de quebra de linha. (Pense nisso como uma correspondência de caracteres de “espaço”.)
\S	Qualquer caractere que *não* seja um espaço, uma tabulação ou uma quebra de linha.

As classes de caracteres são convenientes para reduzir as expressões regulares. A classe de caracteres [0-5] corresponderá somente aos números de 0 a 5; isso é muito mais conciso do que digitar (0|1|2|3|4|5).

Por exemplo, digite o seguinte no shell interativo:

```
>>> xmasRegex = re.compile(r'\d+\s\w+')
>>> xmasRegex.findall('12 drummers, 11 pipers, 10 lords, 9 ladies, 8 maids, 7
swans, 6 geese, 5 rings, 4 birds, 3 hens, 2 doves, 1 partridge')
['12 drummers', '11 pipers', '10 lords', '9 ladies', '8 maids', '7 swans', '6 geese', '5 rings', '4
birds', '3 hens', '2 doves', '1 partridge']
```

A expressão regular \d+\s\w+ corresponderá a textos que tenham um ou mais dígitos (\d+) seguidos de um caractere de espaço em branco (\s) seguido de um ou mais caracteres que sejam letra/dígito/underscore (\w+). O método findall() retorna todas as strings que correspondam ao padrão da regex em uma lista.

Criando suas próprias classes de caracteres

Haverá ocasiões em que você vai querer fazer a correspondência de um conjunto de caracteres, porém as classes abreviadas de caracteres (\d, \w, \s e assim por diante) serão amplas demais. Você pode definir sua própria classe de caracteres usando colchetes. Por exemplo, a classe de caracteres [aeiouAEIOU] corresponderá a qualquer vogal, tanto minúscula quanto maiúscula. Digite o seguinte no shell interativo:

```
>>> vowelRegex = re.compile(r'[aeiouAEIOU]')
>>> vowelRegex.findall('RoboCop eats baby food. BABY FOOD.')
['o', 'o', 'o', 'e', 'a', 'a', 'o', 'o', 'A', 'O', 'O']
```

Também é possível incluir intervalos de letras ou de números usando um hífen. Por exemplo, a classe de caracteres [a-zA-Z0-9] corresponderá a todas as letras minúsculas, às letras maiúsculas e aos números.

Observe que, nos colchetes, os símbolos normais de expressão regular não são interpretados. Isso quer dizer que não é necessário escapar os caracteres .., *, ? ou () com uma barra invertida na frente. Por exemplo, a classe de caracteres [0-5.] corresponderá aos dígitos de 0 a 5 e um ponto. Não é preciso escrever essa classe como [0-5\.].

Ao inserir um acento circunflexo (^) logo depois do colchete de abertura da classe de caracteres, podemos criar uma *classe negativa de caracteres*. Uma classe negativa de caracteres corresponderá a todos os caracteres que *não* estejam na classe de caracteres. Por exemplo, digite o seguinte no shell interativo:

```
>>> consonantRegex = re.compile(r'[^aeiouAEIOU]')
>>> consonantRegex.findall('RoboCop eats baby food. BABY FOOD.')
['R', 'b', 'c', 'p', ' ', 't', 's', ' ', 'b', 'b', 'y', ' ', 'f', 'd', '.', ' ', 'B', 'B', 'Y', ' ', 'F', 'D', '.']
```

Agora, em vez de fazer a correspondência de todas as vogais, estamos fazendo a correspondência de todos os caracteres que não sejam uma vogal.

Acento circunflexo e o sinal de dólar

O símbolo de acento circunflexo (^) também pode ser usado no início de uma regex para indicar que uma correspondência deve ocorrer no *início* de um texto pesquisado. Da mesma maneira, podemos colocar um sinal de dólar ($) no final da regex para indicar que a string deve *terminar* com esse padrão de regex. Além disso, podemos usar ^ e $ juntos para indicar que a string toda deve corresponder à regex – ou seja, não é suficiente que uma correspondência seja feita com algum subconjunto da string.

Por exemplo, a string r'^Hello' de expressão regular corresponde a strings que comecem com 'Hello'. Digite o seguinte no shell interativo:

```
>>> beginsWithHello = re.compile(r'^Hello')
>>> beginsWithHello.search('Hello world!')
<_sre.SRE_Match object; span=(0, 5), match='Hello'>
>>> beginsWithHello.search('He said hello.') == None
True
```

A string r'\d$' de expressão regular corresponde a strings que terminem com um caractere numérico de 0 a 9. Digite o seguinte no shell interativo:

```
>>> endsWithNumber = re.compile(r'\d$')
>>> endsWithNumber.search('Your number is 42')
<_sre.SRE_Match object; span=(16, 17), match='2'>
>>> endsWithNumber.search('Your number is forty two.') == None
True
```

A string r'^\d+$' de expressão regular corresponde a strings que comecem e terminem com um ou mais caracteres numéricos. Digite o seguinte no shell interativo:

```
>>> wholeStringIsNum = re.compile(r'^\d+$')
>>> wholeStringIsNum.search('1234567890')
<_sre.SRE_Match object; span=(0, 10), match='1234567890'>
>>> wholeStringIsNum.search('12345xyz67890') == None
True
>>> wholeStringIsNum.search('12  34567890') == None
True
```

As duas últimas chamadas a search() no exemplo anterior com o shell interativo mostram como a string toda deve corresponder à regex se ^ e $ forem utilizados.

Sempre confundo o significado desses dois símbolos; sendo assim, utilizo o mnemônico “Carrots cost dollars” para me lembrar de que o acento circunflexo (caret) vem antes e o sinal de dólar vem depois.

Caractere-curinga

O caractere . (ou *ponto*) em uma expressão regular é chamado de *caractere-curinga* e corresponde a qualquer caractere, exceto uma quebra de linha. Por exemplo, digite o seguinte no shell interativo:

```
>>> atRegex = re.compile(r'.at')
>>> atRegex.findall('The cat in the hat sat on the flat mat.')
['cat', 'hat', 'sat', 'lat', 'mat']
```

Lembre-se de que o caractere ponto corresponderá somente a um caractere, motivo pelo qual a correspondência para o texto flat no exemplo anterior foi feita somente com lat. Para fazer a correspondência de um ponto propriamente dito, escape-o com uma barra invertida, ou seja, use \..

Correspondendo a tudo usando ponto-asterisco

Às vezes, vamos querer fazer uma correspondência de tudo. Por exemplo, suponha que você queira fazer a correspondência da string 'First Name:' seguida de todo e qualquer texto seguido de 'Last Name:' e, por fim, seguida de qualquer caractere novamente. Podemos usar ponto-asterisco (.*) para indicar "qualquer caractere". Lembre-se de que o caractere ponto quer dizer "qualquer caractere único, exceto a quebra de linha" e o caractere asterisco quer dizer "zero ou mais ocorrências do caractere anterior".

Digite o seguinte no shell interativo:

```
>>> nameRegex = re.compile(r'First Name: (.*) Last Name: (.*)')
>>> mo = nameRegex.search('First Name: Al Last Name: Sweigart')
>>> mo.group(1)
'Al'
>>> mo.group(2)
'Sweigart'
```

O ponto-asterisco utiliza o modo *greedy*: ele sempre tentará fazer a correspondência do máximo de texto possível. Para corresponder a todo e qualquer texto em modo *nongreedy*, utilize ponto, asterisco e ponto de interrogação (.*?). Assim como no caso das chaves, o ponto de interrogação diz ao Python para fazer a correspondência em modo nongreedy.

Digite o seguinte no shell interativo para ver a diferença entre as versões greedy e nongreedy:

```
>>> nongreedyRegex = re.compile(r'<.*?>')
>>> mo = nongreedyRegex.search('<To serve man> for dinner.>')
>>> mo.group()
'<To serve man>'
```

```
>>> greedyRegex = re.compile(r'<.*>')
>>> mo = greedyRegex.search('<To serve man> for dinner.>')
>>> mo.group()
'<To serve man> for dinner.>'
```

Ambas as regexes, de modo geral, podem ser traduzidas como “faça a correspondência de um sinal de ‘menor que’ seguido de qualquer caractere seguido de um sinal de ‘maior que’”. Porém a string '<To serve man> for dinner.>' tem duas correspondências possíveis para o sinal de ‘maior que’. Na versão nongreedy da regex, o Python faz a correspondência com a menor string possível: '<To serve man>'. Na versão greedy, o Python faz a correspondência com a maior string possível: '<To serve man> for dinner.>'.

Correspondendo a quebras de linha com o caractere ponto

O ponto-asterisco corresponderá a qualquer caractere, exceto uma quebra de linha. Ao passar re.DOTALL como segundo argumento de re.compile(), podemos fazer o caractere ponto corresponder a *todos* os caracteres, incluindo o caractere de quebra de linha.

Digite o seguinte no shell interativo:

```
>>> noNewlineRegex = re.compile('.*')
>>> noNewlineRegex.search('Serve the public trust.\nProtect the innocent.
\nUphold the law.').group()
'Serve the public trust.'

>>> newlineRegex = re.compile('.*', re.DOTALL)
>>> newlineRegex.search('Serve the public trust.\nProtect the innocent.
\nUphold the law.').group()
'Serve the public trust.\nProtect the innocent.\nUphold the law.'
```

A regex noNewlineRegex, criada sem que re.DOTALL tenha sido passado para a chamada a re.compile(), corresponderá a qualquer caractere até o primeiro caractere de quebra de linha, enquanto newlineRegex, que *teve* re.DOTALL passado para re.compile(), corresponderá a todos os caracteres. É por isso que a chamada a newlineRegex.search() corresponde à string completa, incluindo seus

caracteres de quebra de linha.

Revisão dos símbolos de regex

Este capítulo discutiu bastante a notação; sendo assim, apresentaremos uma revisão rápida do que aprendemos:

- ? corresponde a zero ou uma ocorrência do grupo anterior.
- * corresponde a zero ou mais ocorrências do grupo anterior.
- + corresponde a uma ou mais ocorrências do grupo anterior.
- {n} corresponde a exatamente *n* ocorrências do grupo anterior.
- {n,} corresponde a *n* ou mais ocorrências do grupo anterior.
- {,m} corresponde a zero até *m* ocorrências do grupo anterior.
- {n,m} corresponde a no mínimo *n* e no máximo *m* ocorrências do grupo anterior.
- {n,m}? ou *? ou +? faz uma correspondência nongreedy do grupo anterior.
- ^spam quer dizer que a string deve começar com *spam*.
- spam$ quer dizer que a string dever terminar com *spam*.
- . corresponde a qualquer caractere, exceto os caracteres de quebra de linha.
- \d, \w e \s correspondem a um dígito, um caractere de palavra ou um caractere de espaço, respectivamente.
- \D, \W e \S correspondem a qualquer caractere, exceto um dígito, um caractere de palavra ou um caractere de espaço, respectivamente.
- [abc] corresponde a qualquer caractere que estiver entre os colchetes (por exemplo, *a*, *b* ou *c*).
- [^abc] corresponde a qualquer caractere que não esteja entre os colchetes.

Correspondências sem diferenciar letras maiúsculas de minúsculas

Normalmente, as expressões regulares fazem correspondência de textos com o tipo exato de letra, ou seja, maiúscula ou minúscula, que você especificar. Por exemplo, as regexes a seguir fazem a correspondência de strings totalmente diferentes:

```
>>> regex1 = re.compile('RoboCop')
>>> regex2 = re.compile('ROBOCOP')
>>> regex3 = re.compile('robOcop')
>>> regex4 = re.compile('RobocOp')
```

Entretanto, às vezes, você estará preocupado somente em fazer a correspondência das letras, sem se importar se elas são maiúsculas ou minúsculas. Para fazer sua regex ignorar as diferenças entre letras maiúsculas e minúsculas (ser case-insensitive), re.IGNORECASE ou re.I pode ser passado como segundo argumento de re.compile(). Digite o seguinte no shell interativo:

```
>>> robocop = re.compile(r'robocop', re.I)
>>> robocop.search('RoboCop is part man, part machine, all cop.').group()
'RoboCop'

>>> robocop.search('ROBOCOP protects the innocent.').group()
'ROBOCOP'

>>> robocop.search('Al, why does your programming book talk about robocop so much?').group()
'robocop'
```

Substituindo strings com o método sub()

As expressões regulares não só podem identificar padrões de texto como também podem substituir esses padrões por novos textos. O método sub() dos objetos Regex recebe dois argumentos. O primeiro argumento é uma string para substituir qualquer correspondência. O segundo é a string para a expressão regular. O método sub() retorna uma string com as substituições aplicadas.

Por exemplo, digite o seguinte no shell interativo:

```
>>> namesRegex = re.compile(r'Agent \w+')
>>> namesRegex.sub('CENSORED', 'Agent Alice gave the secret documents to Agent Bob.')
'CENSORED gave the secret documents to CENSORED.'
```

Às vezes, pode ser necessário utilizar o próprio texto correspondente como parte da substituição. No primeiro argumento de sub(), podemos digitar \1, \2, \3 e assim por diante para dizer "insira o texto do grupo 1, 2, 3 e assim por diante na substituição".

Por exemplo, suponha que você queira censurar os nomes dos agentes secretos mostrando apenas as primeiras letras de seus nomes. Para isso, podemos usar a regex Agent (\w)\w* e passar r'\1****' como o primeiro argumento de sub(). \1 nessa string será substituído por qualquer texto correspondente no grupo 1 – ou seja, o grupo (\w) da expressão regular.

```
>>> agentNamesRegex = re.compile(r'Agent (\w)\w*')
>>> agentNamesRegex.sub(r'\1****', 'Agent Alice told Agent Carol that Agent Eve knew Agent Bob was a double agent.')
A**** told C**** that E**** knew B**** was a double agent.'
```

Administrando regexes complexas

As expressões regulares serão convenientes se o padrão de texto para a correspondência for simples. Porém fazer a correspondência de padrões complicados de texto pode exigir expressões regulares longas e confusas. Podemos atenuar esse problema dizendo à função re.compile() que ignore espaços em branco e comentários na string de expressão regular. Esse "modo verbose" pode ser habilitado se a variável re.VERBOSE for passada como segundo argumento de re.compile().

Agora, em vez de uma expressão regular difícil de ler como:

```
phoneRegex = re.compile(r'((\d{3}|\(\d{3}\))?(\s|-|\.)?\d{3}(\s|-|\.)\d{4}(\s*
(ext|x|ext.)\s*\d{2,5})?)')
```

podemos distribuir a expressão regular em várias linhas usando comentários como:

```
phoneRegex = re.compile(r'''(
    (\d{3}|\(\d{3}\))?          # código de área
    (\s|-|\.)?                  # separador
    \d{3}                       # primeiros 3 dígitos
    (\s|-|\.)                   # separador
    \d{4}                       # últimos 4 dígitos
```

```
(\s*(ext|x|ext.)\s*\d{2,5})?  # extensão
)''', re.VERBOSE)
```

Observe como o exemplo anterior utiliza a sintaxe de aspas triplas (''') para criar uma string de múltiplas linhas de modo que a definição da expressão regular possa ser distribuída em diversas linhas, tornando-a muito mais legível.

As regras para comentários em uma string de expressão regular são as mesmas usadas no código Python normal: o símbolo # e tudo que estiver depois dele até o final da linha serão ignorados. Além disso, os espaços extras na string de múltiplas linhas da expressão regular não serão considerados como parte do padrão de texto para a correspondência. Isso permite organizar a expressão regular para que ela se torne mais fácil de ler.

Combinando re.IGNORECASE, re.DOTALL e re.VERBOSE

O que aconteceria se você quisesse usar re.VERBOSE para escrever comentários em sua expressão regular, mas também quisesse utilizar re.IGNORECASE para ignorar as diferenças entre letras maiúsculas e minúsculas? Infelizmente, a função re.compile() aceita apenas um único valor como segundo argumento. Podemos contornar essa limitação combinando as variáveis re.IGNORECASE, re.DOTALL e re.VERBOSE utilizando o caractere pipe (|) que, nesse contexto, é conhecido como o operador *ou bit a bit* (bitwise or).

Portanto, se quiser uma expressão regular que ignore as diferenças entre letras maiúsculas e minúsculas *e* inclua quebras de linha para que correspondam ao caractere ponto, sua chamada a re.compile() deverá ser feita da seguinte forma:

```
>>> someRegexValue = re.compile('foo', re.IGNORECASE | re.DOTALL)
```

As três opções para o segundo argumento terão o aspecto a seguir:

```
>>> someRegexValue = re.compile('foo', re.IGNORECASE | re.DOTALL |
re.VERBOSE)
```

Essa sintaxe é um pouco antiga e tem origem nas primeiras versões do Python. Os detalhes sobre os operadores bit a bit estão além do escopo deste livro, porém dê uma olhada nos recursos em *http://nostarch.com/automatestuff/* para obter mais informações. Também podemos passar outras opções para o segundo argumento; elas são incomuns, porém você poderá ler mais sobre elas também no site de recursos.

Projeto: extrator de números de telefone e de endereços de email

Suponha que você tenha a tarefa maçante de localizar todos os números de telefone e endereços de email em uma página web ou um documento extenso. Se fizer rolagens manualmente pela página, você poderá acabar fazendo a pesquisa por bastante tempo. Porém, se você tivesse um programa que pudesse pesquisar o texto em seu clipboard em busca de números de telefone e de endereços de email, seria possível simplesmente pressionar CTRL-A para selecionar todo o texto, CTRL-C para copiá-lo para o clipboard e então executar o seu programa. Ele poderia substituir o texto no clipboard somente pelos números de telefone e pelos endereços de email encontrados.

Sempre que estiver diante de um novo projeto, pode ser tentador mergulhar diretamente na escrita do código. No entanto, com muita frequência, será melhor dar um passo para trás e considerar o quadro geral. Recomendo inicialmente definir um plano geral para o que seu programa deverá fazer. Não pense ainda no código propriamente dito – você poderá se preocupar com ele depois. Neste momento, atenha-se aos aspectos mais gerais.

Por exemplo, seu extrator de números de telefone e de endereços de email deverá fazer o seguinte:

- Obter o texto do clipboard.
- Encontrar todos os números de telefone e os endereços de email no texto.
- Colá-los no clipboard.

Agora você poderá começar a pensar em como isso funcionará no código. O código deverá fazer o seguinte:

- Usar o módulo `pyperclip` para copiar e colar strings.

- Criar duas regexes: uma para corresponder a números de telefone e outra para endereços de email.
- Encontrar todas as correspondências, e não apenas a primeira, para ambas as regexes.
- Formatar as strings correspondentes de forma elegante em uma única string a ser colada no clipboard.
- Exibir algum tipo de mensagem caso nenhuma correspondência tenha sido encontrada no texto.

Essa lista é como um roadmap (mapa) do projeto. À medida que escrever o código, você poderá focar em cada um desses passos separadamente. Cada passo é razoavelmente administrável e está expresso em termos de tarefas que você já sabe fazer em Python.

Passo 1: Criar uma regex para números de telefone

Inicialmente, você deve criar uma expressão regular para procurar números de telefone. Crie um arquivo novo, digite o código a seguir e salve-o como *phoneAndEmail.py*:

```
#! python3
# phoneAndEmail.py – Encontra números de telefone e endereços de email no clipboard.

import pyperclip, re

phoneRegex = re.compile(r'''(
    (\d{3}|\(\d{3}\))?                # código de área
    (\s|-|\.)?                        # separador
    (\d{3})                           # primeiros 3 dígitos
    (\s|-|\.)                         # separador
    (\d{4})                           # últimos 4 dígitos
    (\s*(ext|x|ext.)\s*(\d{2,5}))?    # extensão
    )''', re.VERBOSE)

# TODO: Cria a regex para email.

# TODO: Encontra correspondências no texto do clipboard.

# TODO: Copia os resultados para o clipboard.
```

Os comentários TODO são apenas um esqueleto para o programa. Eles serão substituídos à medida que você escrever o código propriamente dito.

O número de telefone começa com um código de área *opcional*, portanto o grupo para código de área é seguido de um ponto de interrogação. Como o código de área pode ter apenas três dígitos (isto é, \d{3}) *ou* três dígitos entre parênteses (isto é, \(\d{3}\)), deve haver um pipe unindo essas partes. Você pode adicionar o comentário # código de área na regex para essa parte da string de múltiplas linhas; isso ajudará a se lembrar a que (\d{3}|\(\d{3}\))? deve corresponder.

O caractere separador do número de telefone pode ser um espaço (\s), um hífen (-) ou um ponto (.), portanto essas partes também devem ser unidas por pipes. As próximas partes da expressão regular são simples: três dígitos seguidos de outro separador seguido de quatro dígitos. A última parte é uma extensão opcional composta de qualquer quantidade de espaços seguida de ext, x ou ext. seguida de dois a cinco dígitos.

Passo 2: Criar uma regex para endereços de email

Também será necessário ter uma expressão regular que possa corresponder a endereços de email. Faça seu programa ter o seguinte aspecto:

```
#! python3
# phoneAndEmail.py – Encontra números de telefone e endereços de email no clipboard.

import pyperclip, re

phoneRegex = re.compile(r'''(
--trecho removido--

# Cria regex para email.
emailRegex = re.compile(r'''(
u    [a-zA-Z0-9._%+-]+      # nome do usuário
v    @                      # símbolo @
w    [a-zA-Z0-9.-]+         # nome do domínio
     (\.[a-zA-Z]{2,4})      # ponto seguido de outros caracteres
     )''', re.VERBOSE)

# TODO: Encontra correspondências no texto do clipboard.
```

```
# TODO: Copia os resultados para o clipboard.
```

A parte referente ao nome do usuário no endereço de email u tem um ou mais caracteres que podem ser: letras maiúsculas e minúsculas, números, um ponto, um underscore, um sinal de porcentagem, um sinal de adição ou um hífen. Tudo isso pode ser colocado em uma classe de caracteres: [a-zA-Z0-9._%+-].

O domínio e o nome do usuário são separados por um símbolo de @ v. O nome de domínio w tem uma classe de caracteres um pouco menos permissiva, contendo apenas letras, números, pontos e hifens: [a-zA-Z0-9.-]. Por fim, temos a parte "ponto com" (tecnicamente conhecida como *domínio de mais alto nível*) que, na verdade, pode ser um ponto seguido de qualquer caractere. Essa parte tem entre dois e quatro caracteres.

O formato dos endereços de email tem muitas regras singulares. Essa expressão regular não corresponderá a todos os endereços possíveis de email, porém corresponderá a quase todos os endereços típicos de email que você encontrar.

Passo 3: Encontrar todas as correspondências no texto do clipboard

Agora que especificamos as expressões regulares para números de telefone e endereços de email, podemos deixar o módulo re do Python fazer o trabalho pesado de localizar todas as correspondências no clipboard. A função pyperclip.paste() obterá um valor de string com o texto que está no clipboard e o método de regex findall() retornará uma lista de tuplas.

Faça seu programa ter o seguinte aspecto:

```
#! python3
# phoneAndEmail.py – Encontra números de telefone e endereços de email no clipboard.

import pyperclip, re

phoneRegex = re.compile(r'''(
--trecho removido--
```

```
   # Encontra as correspondências no texto do clipboard.
   text = str(pyperclip.paste())

u matches = []
v for groups in phoneRegex.findall(text):
       phoneNum = '-'.join([groups[1], groups[3], groups[5]])
       if groups[8] != '':
          phoneNum += ' x' + groups[8]
       matches.append(phoneNum)
w for groups in emailRegex.findall(text):
       matches.append(groups[0])
   # TODO: Copia os resultados para o clipboard.
```

Há uma tupla para cada correspondência e cada tupla contém strings para cada grupo da expressão regular. Lembre-se de que o grupo 0 corresponde à expressão regular completa, portanto o grupo no índice 0 da tupla é aquele em que estaremos interessados.

Como podemos ver em u, as correspondências serão armazenadas em uma variável de lista chamada matches. O programa começa com uma lista vazia e dois loops for. Para os endereços de email, devemos concatenar o grupo 0 de cada correspondência w. Para os números de telefone correspondentes, não queremos concatenar somente o grupo 0. Embora o programa *detecte* números de telefone em diversos formatos, queremos que esse número seja concatenado em um formato único e padrão. A variável phoneNum contém uma string criada a partir dos grupos 1, 3, 5 e 8 do texto correspondente v. (Esses grupos são: o código de área, os três primeiros dígitos, os quatro últimos dígitos e a extensão.)

Passo 4: Reunir as correspondências em uma string para o clipboard

Agora que temos os endereços de email e os números de telefone na forma de uma lista de strings em matches, devemos inseri-los no clipboard. A função pyperclip.copy() aceita apenas um único valor de string, e não uma lista de strings; sendo assim, você deve chamar o método join() em matches.

Para fazer com que seja mais fácil ver que o programa está funcionando, vamos exibir qualquer correspondência encontrada no terminal. Se nenhum

número de telefone ou endereço de email for encontrado, o programa deverá informar isso ao usuário.

Faça seu programa ter o seguinte aspecto:

```
#! python3
# phoneAndEmail.py – Encontra números de telefone e endereços de email no clipboard.

--trecho removido--
for groups in emailRegex.findall(text):
    matches.append(groups[0])

# Copia os resultados para o clipboard.
if len(matches) > 0:
    pyperclip.copy('\n'.join(matches))
    print('Copied to clipboard:')
    print('\n'.join(matches))
else:
    print('No phone numbers or email addresses found.')
```

Executando o programa

Para ver um exemplo, abra seu navegador web na página de contato da No Starch Press em *http://www.nostarch.com/contactus.htm*, tecle CTRL-A para selecionar todo o texto da página e CTRL-C para copiá-lo para o clipboard. Ao executar esse programa, a saída será semelhante a:

```
Copied to clipboard:
800-420-7240
415-863-9900
415-863-9950
info@nostarch.com
media@nostarch.com
academic@nostarch.com
help@nostarch.com
```

Ideias para programas semelhantes

Identificar padrões de texto (e possivelmente substituí-los com o método sub()) tem várias aplicações diferentes em potencial.

• Encontrar URLs de sites que comecem com *http://* ou com *https://*.

- Limpar datas em formatos diferentes (como 3/14/2015, 03-14-2015 e 2015/3/14) substituindo-as por datas em um único formato-padrão.
- Remover informações críticas como números de seguridade social (Social Security Number) ou números de cartões de crédito.
- Encontrar erros comuns de digitação como vários espaços entre palavras, repetição acidental acidental de palavras ou vários pontos de exclamação no final das sentenças. São irritantes!!

Resumo

Embora um computador possa procurar um texto rapidamente, devemos dizer-lhe exatamente o que deverá ser procurado. As expressões regulares permitem especificar padrões exatos de caracteres que estivermos procurando. Com efeito, alguns processadores de texto e aplicativos de planilhas oferecem funcionalidades para pesquisar e substituir que permitem fazer pesquisas usando expressões regulares.

O módulo re que acompanha o Python permite compilar objetos Regex. Esses valores têm diversos métodos: search() para encontrar uma única correspondência, findall() para encontrar todas as instâncias correspondentes e sub() para pesquisar e substituir texto.

Há mais sobre a sintaxe de expressões regulares do que foi descrito neste capítulo. Você poderá descobrir mais informações na documentação oficial do Python em *http://docs.python.org/3/library/re.html*. O site de tutoriais em *http://www.regular-expressions.info/* também é um recurso útil.

Agora que você tem expertise para manipular e fazer correspondência de strings, é hora de mergulhar de cabeça na leitura e na escrita de arquivos no disco rígido de seu computador.

Exercícios práticos

1. Qual é a função que cria objetos Regex?
2. Por que as strings puras (raw strings) geralmente são usadas na criação de objetos Regex?

3. O que o método search() retorna?
4. Como podemos obter as strings correspondentes ao padrão a partir de um objeto Match?
5. Na regex criada a partir de r'(\d\d\d)-(\d\d\d-\d\d\d\d)', o que o grupo 0 inclui? E o grupo 1? E o grupo 2?
6. Os parênteses e os pontos têm significados específicos na sintaxe das expressões regulares. Como podemos especificar uma regex que corresponda aos caracteres que representam parênteses e pontos?
7. O método findall() retorna uma lista de strings ou uma lista de tuplas de strings. O que faz com que uma ou outra opção seja retornada?
8. O que o caractere | representa em expressões regulares?
9. Quais são os dois significados do caractere ? em expressões regulares?
10. Qual é a diferença entre os caracteres + e * em expressões regulares?
11. Qual é a diferença entre {3} e {3,5} em expressões regulares?
12. O que as classes abreviadas de caracteres \d, \w e \s representam em expressões regulares?
13. O que as classes abreviadas de caracteres \D, \W e \S representam em expressões regulares?
14. Como podemos fazer uma expressão regular ignorar as diferenças entre letras maiúsculas e minúsculas (ser case-insensitive)?
15. A que o caractere . normalmente corresponde? A que ele corresponderá se re.DOTALL for passado como segundo argumento de re.compile()?
16. Qual é a diferença entre .* e .*??
17. Qual é a sintaxe da classe de caracteres que corresponde a todos os números e a todas as letras minúsculas?
18. Se numRegex = re.compile(r'\d+'), o que numRegex.sub('X', '12 drummers, 11 pipers, five rings, 3 hens') retornará?
19. O que a especificação de re.VERBOSE como segundo argumento de re.compile() permite fazer?
20. Como você poderá escrever uma regex que corresponda a um número com vírgulas a cada três dígitos? Ela deverá corresponder a:
   - '42'

- '1,234'
- '6,368,745'

mas não a:

- '12,34,567' (que tem somente dois dígitos entre as vírgulas)
- '1234' (que não tem vírgulas)

21. Como você poderá escrever uma regex que corresponda ao nome completo de alguém cujo sobrenome seja Nakamoto? Suponha que o primeiro nome que vem antes dele sempre seja uma única palavra que comece com uma letra maiúscula. A regex deverá corresponder a:

- 'Satoshi Nakamoto'
- 'Alice Nakamoto'
- 'RoboCop Nakamoto'

mas não a:

- 'satoshi Nakamoto' (em que o primeiro nome não começa com letra maiúscula)
- 'Mr. Nakamoto' (em que a palavra anterior tem um caractere que não é uma letra)
- 'Nakamoto' (que não tem primeiro nome)
- 'Satoshi nakamoto' (em que Nakamoto não começa com letra maiúscula)

22. Como você poderá escrever uma regex que corresponda a uma frase em que a primeira palavra seja *Alice*, *Bob* ou *Carol*, a segunda palavra seja *eats*, *pets* ou *throws*, a terceira palavra seja *apples*, *cats* ou *baseballs* e a frase termine com um ponto? Essa regex não deve diferenciar letras maiúsculas de minúsculas. Ela deverá corresponder a:

- 'Alice eats apples.'
- 'Bob pets cats.'
- 'Carol throws baseballs.'
- 'Alice throws Apples.'
- 'BOB EATS CATS.'

mas não a:

- 'RoboCop eats apples.'

- 'ALICE THROWS FOOTBALLS.'
- 'Carol eats 7 cats.'

Projetos práticos

Para exercitar, escreva programas que executem as tarefas a seguir.

Detecção de senhas robustas

Crie uma função que utilize expressões regulares para garantir que a string de senha recebida seja robusta. Uma senha robusta deve ter pelo menos oito caracteres, deve conter tanto letras maiúsculas quanto letras minúsculas e ter pelo menos um dígito. Talvez seja necessário testar a string em relação a diversos padrões de regex para validar sua robustez.

Versão de strip() usando regex

Crie uma função que receba uma string e faça o mesmo que o método de string `strip()`. Se nenhum outro argumento for passado além da string em que a remoção será feita, os caracteres de espaço em branco serão removidos do início e do fim da string. Caso contrário, os caracteres especificados no segundo argumento da função serão removidos da string.

---

1 N.T.: Tradução literal de acordo com a citação original em inglês: “Knowing [regular expressions] can mean the difference between solving a problem in 3 steps and solving it in 3,000 steps. When you’re a nerd, you forget that the problems you solve with a couple keystrokes can take other people days of tedious, error-prone work to slog through.”

2 Cory Doctorow, “Here’s what ICT should really teach kids: how to do regular expressions” (Eis o que o ICT realmente deveria ensinar às crianças: como criar expressões regulares), *Guardian*, 4 de dezembro de 2012, *http://www.theguardian.com/technology/2012/dec/04/ict-teach-kids-regular-expressions/*.

CAPÍTULO 8

LENDO E ESCREVENDO EM ARQUIVOS

As variáveis são uma maneira conveniente de armazenar dados enquanto seu programa estiver executando, porém, se quiser que seus dados persistam mesmo após o programa ter encerrado, será necessário salvá-los em um arquivo. Podemos pensar no conteúdo de um arquivo como um único valor de string, potencialmente com gigabytes de tamanho. Neste capítulo, aprenderemos a usar o Python para criar, ler e salvar arquivos no disco rígido.

Arquivos e paths de arquivo

Um arquivo tem duas propriedades fundamentais: um *nome de arquivo* e um *path*. O path especifica a localização de um arquivo no computador. Por exemplo, há um arquivo em meu laptop com Windows 7 cujo nome é *projects.docx* no path *C:\Users\asweigart\Documents*. A parte do nome do arquivo após o último ponto é chamada de *extensão* do arquivo e informa o seu tipo. *project.docx* é um documento Word, e *Users*, *asweigart* e *Documents* referem-se a *pastas* (também chamadas de *diretórios*). As pastas podem conter arquivos e outras pastas. Por exemplo, *project.docx* está na pasta *Documents* que está na pasta *asweigart* que está na pasta *Users*. A figura 8.1 mostra a organização dessa pasta.

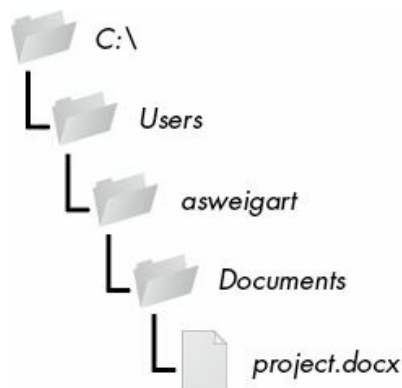



Figura 8.1 – Um arquivo em uma hierarquia de pastas.

A parte referente a *C:* do path é a *pasta-raiz*, que contém todas as demais pastas. No Windows, a pasta-raiz chama-se *C:* e é também chamada de *drive C:*. No OS X e no Linux, a pasta-raiz é */*. Neste livro, usarei a pasta-raiz *C:* no estilo do Windows. Se você estiver inserindo os exemplos do shell interativo no OS X ou no Linux, especifique */* no lugar de *C:*.

Volumes adicionais, como um drive de DVD ou de pen drive USB, aparecerão de forma diferente em sistemas operacionais distintos. No Windows, eles aparecerão como drives-raiz com uma nova letra, por exemplo, *D:* ou *E:*. No OS X, aparecerão como novas pastas dentro da pasta */Volumes*. No Linux, aparecerão como novas pastas dentro da pasta */mnt* ("mount"). Além disso, observe que, enquanto os nomes das pastas e dos arquivos não fazem distinção entre letras maiúsculas e minúsculas no Windows e no OS X, essa diferença existe no Linux.

Barra invertida no Windows e barra para frente no OS X e no Linux

No Windows, os paths são escritos com barras invertidas (\) como separador entre os nomes das pastas. No OS X e no Linux, porém, utilize a barra para frente (/) como separador de path. Se quiser que seus programas funcionem em todos os sistemas operacionais, seus scripts Python deverão ser escritos para tratar ambos os casos.

Felizmente, isso é simples de fazer com a função `os.path.join()`. Se você lhe passar os valores de string com os nomes individuais dos arquivos e das pastas de seu path, `os.path.join()` retornará uma string com um path de arquivo que utilizará os separadores corretos de path. Digite o seguinte no shell interativo:

```
>>> import os
>>> os.path.join('usr', 'bin', 'spam')
'usr\\bin\\spam'
```

Estou executando esses exemplos de shell interativo no Windows, portanto `os.path.join('usr', 'bin', 'spam')` retornou `'usr\\bin\\spam'`. (Observe que as barras invertidas estão duplicadas, pois cada barra invertida deve ser escapada por outro caractere de barra invertida.) Se essa função tivesse sido chamada no

OS X ou no Linux, a string seria 'usr/bin/spam'.

A função os.path.join() será útil se houver necessidade de criar strings para os nomes de arquivo. Essas strings serão passadas para diversas funções relacionadas a arquivos que serão apresentadas neste capítulo. O exemplo a seguir une os nomes de uma lista de nomes de arquivo no final do nome de uma pasta:

```
>>> myFiles = ['accounts.txt', 'details.csv', 'invite.docx']
>>> for filename in myFiles:
        print(os.path.join('C:\\Users\\asweigart', filename))
C:\Users\asweigart\accounts.txt
C:\Users\asweigart\details.csv
C:\Users\asweigart\invite.docx
```

Diretório de trabalho atual

Todo programa executado em seu computador tem um *diretório de trabalho atual* (current working directory, ou cwd). Supõe-se que qualquer nome de arquivo ou path que não comece com a pasta-raiz esteja no diretório de trabalho atual. Podemos obter o diretório de trabalho atual como um valor de string usando a função os.getcwd() e alterá-lo com os.chdir(). Digite o seguinte no shell interativo:

```
>>> import os
>>> os.getcwd()
'C:\\Python34'
>>> os.chdir('C:\\Windows\\System32')
>>> os.getcwd()
'C:\\Windows\\System32'
```

Nesse caso, o diretório de trabalho atual está definido como *C:\Python34*, portanto o nome de arquivo *project.docx* refere-se a *C:\Python34\project.docx*. Quando alteramos o diretório de trabalho atual para *C:\Windows*, *project.docx* é interpretado como *C:\Windows\project.docx*.

O Python exibirá um erro se você tentar mudar para um diretório inexistente.

```
>>> os.chdir('C:\\ThisFolderDoesNotExist')
```

```
Traceback (most recent call last):
  File "<pyshell#18>", line 1, in <module>
    os.chdir('C:\\ThisFolderDoesNotExist')
FileNotFoundError: [WinError 2] The system cannot find the file specified:
'C:\\ThisFolderDoesNotExist'
```

NOTA Embora pasta seja o nome mais moderno para diretório, observe que o *diretório de trabalho atual* (ou somente *diretório de trabalho*) é o termo-padrão, e não pasta de trabalho atual.

Comparação entre paths absolutos e relativos

Há duas maneiras de especificar um path de arquivo.

- Um *path absoluto*, que sempre começa com a pasta-raiz.
- Um *path relativo*, que é relativo ao diretório de trabalho atual do programa.

Temos também as pastas *ponto* (.) e *ponto-ponto* (..). Essas não são pastas de verdade, mas nomes especiais que podem ser usados em um path. Um único ponto para um nome de pasta é a forma abreviada de "este diretório". Dois pontos (ponto-ponto) quer dizer "a pasta pai".

A figura 8.2 contém um exemplo de algumas pastas e arquivos. Quando o diretório de trabalho atual estiver definido com *C:\bacon*, os paths relativos das outras pastas e dos arquivos serão definidos como mostra a figura.

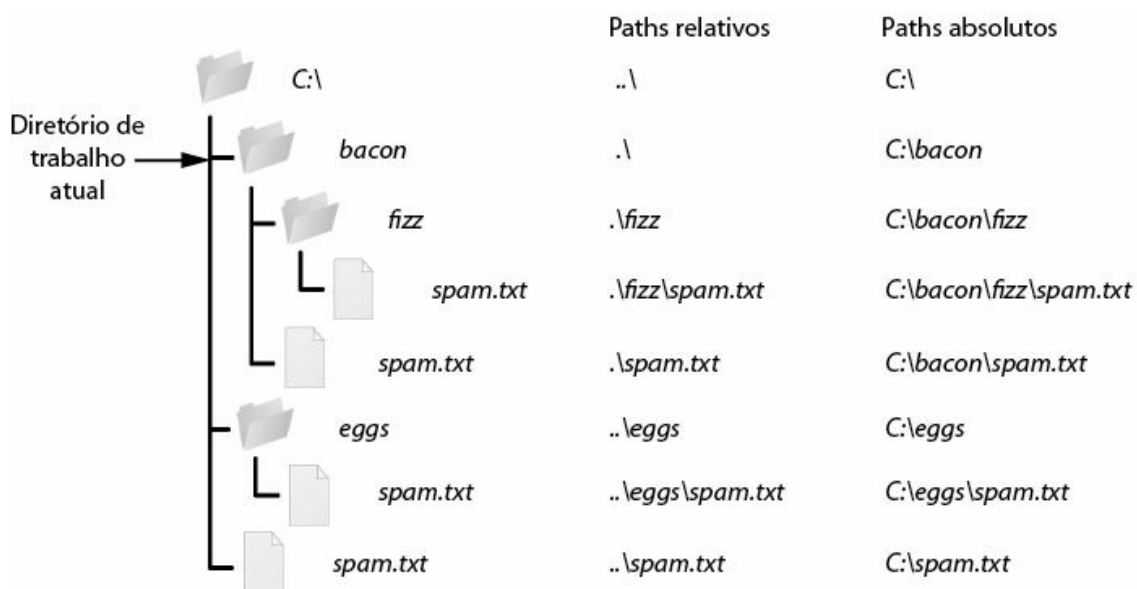



Figura 8.2 – Os paths relativos para as pastas e os arquivos no diretório de trabalho C:\bacon.

O .\ no início de um path relativo é opcional. Por exemplo, *.\spam.txt* e *spam.txt* referem-se ao mesmo arquivo.

Criando novas pastas com os.makedirs()

Seus programas podem criar novas pastas (diretórios) com a função os.makedirs(). Digite o seguinte no shell interativo:

```
>>> import os
>>> os.makedirs('C:\\delicious\\walnut\\waffles')
```

Esse comando criará não só a pasta *C:\delicious* como também uma pasta *walnut* em *C:\delicious* e uma pasta *waffles* em *C:\delicious\walnut*, isto é, os.makedirs() criará qualquer pasta intermediária necessária para garantir que o path completo exista. A figura 8.3 mostra essa hierarquia de pastas.

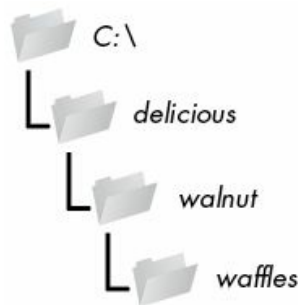



Figura 8.3 – O resultado de os.makedirs('C:\\delicious\\walnut\\waffles').

Módulo os.path

O módulo os.path contém muitas funções úteis relacionadas a nomes e paths de arquivo. Por exemplo, já usamos os.path.join() para criar paths de maneira que isso funcione em qualquer sistema operacional. Como os.path é um módulo do módulo os, podemos importá-lo simplesmente executando import os. Sempre que seus programas precisarem trabalhar com arquivos, pastas ou paths de arquivo, você poderá consultar os pequenos exemplos desta seção. A documentação completa do módulo os.path está no site do Python em *http://docs.python.org/3/library/os.path.html*.

NOTA A maioria dos próximos exemplos nesta seção exigirá o módulo os, portanto lembre-se de importá-lo no início de qualquer script que você criar ou sempre que

reiniciar o IDLE. Caso contrário, uma mensagem de erro NameError: name 'os' is not defined (NameError: nome 'os' não está definido) será exibida.

Lidando com paths absolutos e relativos

O módulo os.path disponibiliza funções que retornam o path absoluto de um path relativo e para verificar se um dado path representa um path absoluto.

- Chamar os.path.abspath(*path*) retornará uma string com o path absoluto referente ao argumento. Essa é uma maneira fácil de converter um path relativo em um path absoluto.
- Chamar os.path.isabs(*path*) retornará True se o argumento for um path absoluto e False se for um path relativo.
- Chamar os.path.relpath(*path*, *início*) retornará uma string contendo um path relativo ao path *início* para *path*. Se *início* não for especificado, o diretório de trabalho atual será usado como path de início.

Teste essas funções no shell interativo:

```
>>> os.path.abspath('.')
'C:\\Python34'
>>> os.path.abspath('.\\Scripts')
'C:\\Python34\\Scripts'
>>> os.path.isabs('.')
False
>>> os.path.isabs(os.path.abspath('.'))
True
```

Como *C:\Python34* era o diretório de trabalho quando os.path.abspath() foi chamado, a pasta "ponto" representa o path absoluto 'C:\\Python34'.

NOTA Como seu sistema provavelmente tem arquivos e pastas diferentes em relação ao meu sistema, você não poderá seguir exatamente todos os exemplos deste capítulo. Apesar disso, tente acompanhar usando pastas que existam em seu computador.

Digite as seguintes chamadas a os.path.relpath() no shell interativo:

```
>>> os.path.relpath('C:\\Windows', 'C:\\')
'Windows'
>>> os.path.relpath('C:\\Windows', 'C:\\spam\\eggs')
'..\\..\\Windows'
>>> os.getcwd()
```

'C:\\Python34'

Chamar os.path.dirname(*path*) retornará uma string contendo tudo que estiver antes da última barra no argumento path. Chamar os.path.basename(*path*) retornará uma string contendo tudo que estiver após a última barra no argumento path. O nome do diretório e o nome base de um path estão representados na figura 8.4.

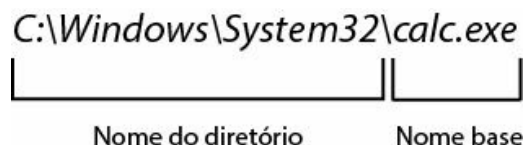



Figura 8.4 – O nome base está depois da última barra em um path e corresponde ao nome do arquivo. O nome do diretório corresponde a tudo que estiver antes da última barra.

Por exemplo, digite o seguinte no shell interativo:

```
>>> path = 'C:\\Windows\\System32\\calc.exe'
>>> os.path.basename(path)
'calc.exe'
>>> os.path.dirname(path)
'C:\\Windows\\System32'
```

Se o nome de diretório e o nome base de um path forem necessários ao mesmo tempo, você poderá simplesmente chamar os.path.split() para obter um valor de tupla contendo essas duas strings, como em:

```
>>> calcFilePath = 'C:\\Windows\\System32\\calc.exe'
>>> os.path.split(calcFilePath)
('C:\\Windows\\System32', 'calc.exe')
```

Observe que a mesma tupla poderia ter sido criada por meio da chamada a os.path.dirname() e a os.path.basename(), com seus valores de retorno inseridos em uma tupla.

```
>>> (os.path.dirname(calcFilePath), os.path.basename(calcFilePath))
('C:\\Windows\\System32', 'calc.exe')
```

Porém os.path.split() será um atalho conveniente se você precisar de ambos os valores.

Além disso, observe que os.path.split() *não* recebe um path de arquivo e retorna uma lista de strings com cada pasta. Para isso, utilize o método de

string split() e separe a string em os.sep. Lembre-se do que vimos anteriormente, que a variável os.sep é definida com a barra correta de separação de pastas para o computador que estiver executando o programa.

Por exemplo, digite o seguinte no shell interativo:

```
>>> calcFilePath.split(os.path.sep)
['C:', 'Windows', 'System32', 'calc.exe']
```

Em sistemas OS X e Linux, haverá uma string vazia no início da lista retornada:

```
>>> '/usr/bin'.split(os.path.sep)
['', 'usr', 'bin']
```

O método de string split() funciona retornando uma lista contendo cada parte do path. Esse método funcionará em qualquer sistema operacional se os.path.sep lhe for passado.

Obtendo os tamanhos dos arquivos e o conteúdo das pastas

Após ter dominado as maneiras de lidar com paths de arquivo, você poderá começar a reunir informações sobre arquivos e pastas específicos. O módulo os.path disponibiliza funções para obter o tamanho de um arquivo em bytes e os arquivos e as pastas que estiverem em uma determinada pasta.

- Chamar os.path.getsize(*path*) retornará o tamanho em bytes do arquivo no argumento *path*.
- Chamar os.listdir(*path*) retornará uma lista de strings com nomes de arquivo para cada arquivo no argumento *path*. (Observe que essa função está no módulo os, e não em os.path.)

Eis o que obtive quando testei essas funções no shell interativo:

```
>>> os.path.getsize('C:\\Windows\\System32\\calc.exe')
776192
>>> os.listdir('C:\\Windows\\System32')
['0409', '12520437.cpx', '12520850.cpx', '5U877.ax', 'aaclient.dll',
--trecho removido--
'xwtpdui.dll', 'xwtpw32.dll', 'zh-CN', 'zh-HK', 'zh-TW', 'zipfldr.dll']
```

Como podemos ver, o programa *calc.exe* em meu computador tem 776.192 bytes e tenho muitos arquivos em *C:\Windows\system32*. Se quiser descobrir o tamanho total de todos os arquivos nesse diretório, poderei usar os.path.getsize() e os.listdir() juntos.

```
>>> totalSize = 0
>>> for filename in os.listdir('C:\\Windows\\System32'):
      totalSize = totalSize + os.path.getsize(os.path.join('C:\\Windows\\System32',
filename))

>>> print(totalSize)
1117846456
```

À medida que percorro todos os nomes de arquivo da pasta *C:\Windows\System32* em um loop, a variável totalSize é incrementada com o tamanho de cada arquivo. Observe que, quando chamo os.path.getsize(), utilizo os.path.join() para unir o nome da pasta ao nome do arquivo atual. O inteiro que os.path.getsize() retorna é somado ao valor de totalSize. Após percorrer todos os arquivos no loop, apresento totalSize para ver o tamanho total da pasta *C:\Windows\System32*.

Verificando a validade de um path

Muitas funções Python falharão gerando um erro se você lhes fornecer um path inexistente. O módulo os.path disponibiliza funções para verificar se um dado path existe e se é um arquivo ou uma pasta.

- Chamar os.path.exists(*path*) retornará True se o arquivo ou a pasta referenciada no argumento existir e retornará False caso contrário.
- Chamar os.path.isfile(*path*) retornará True se o argumento referente ao path existir e for um arquivo e retornará False caso contrário.
- Chamar os.path.isdir(*path*) retornará True se o argumento referente ao path existir e for uma pasta e retornará False caso contrário.

Eis o que obtive quando testei essas funções no shell interativo:

```
>>> os.path.exists('C:\\Windows')
True
```

```
>>> os.path.exists('C:\\some_made_up_folder')
False
>>> os.path.isdir('C:\\Windows\\System32')
True
>>> os.path.isfile('C:\\Windows\\System32')
False
>>> os.path.isdir('C:\\Windows\\System32\\calc.exe')
False
>>> os.path.isfile('C:\\Windows\\System32\\calc.exe')
True
```

Podemos determinar se há um DVD ou um pen drive conectado ao computador no momento fazendo a verificação com a função os.path.exists(). Por exemplo, se eu quiser verificar se há um pen drive com um volume chamado *D:* em meu computador Windows, posso fazer isso com:

```
>>> os.path.exists('D:\\')
False
```

Opa! Parece que me esqueci de conectar o meu pen drive.

Processo de leitura/escrita

Depois que se sentir à vontade para trabalhar com pastas e paths relativos, você poderá especificar a localização dos arquivos a serem lidos e escritos. As funções discutidas nas próximas seções se aplicam a arquivos em formato texto simples. *Arquivos em formato texto simples* (plaintext files) contêm somente caracteres básicos de texto e não incluem informações sobre fonte, tamanho ou cor. Os arquivos-texto com extensão *.txt* ou arquivos de scripts Python com extensão *.py* são exemplos de arquivos em formato texto simples. Eles podem ser abertos com o aplicativo Notepad do Windows ou com o TextEdit do OS X. Seus programas poderão ler facilmente o conteúdo de arquivos em formato texto simples e tratá-los como um valor normal de string.

Os *arquivos binários* correspondem a todos os demais tipos de arquivos, por exemplo, documentos de processadores de texto, PDFs, imagens, planilhas e programas executáveis. Se você abrir um arquivo binário no Notepad ou no TextEdit, seu conteúdo parecerá uma confusão sem sentido,

como mostra a figura 8.5.

Figura 8.5 – O programa calc.exe *do Windows aberto no Notepad.*

Como cada tipo de arquivo binário diferente deve ter tratado de uma maneira própria, este livro não abordará a leitura e a escrita direta de arquivos binários puros. Felizmente, muitos módulos facilitam trabalhar com arquivos binários – você explorará um deles, o módulo shelve, mais adiante neste capítulo.

Em Python, há três passos para ler e escrever em arquivos:

1. Chamar a função open() para que um objeto File seja retornado.
2. Chamar o método read() ou write() no objeto File.
3. Fechar o arquivo chamando o método close() no objeto File.

Abrindo arquivos com a função open()

Para abrir um arquivo com a função open(), passe um path em forma de string indicando o arquivo que você deseja abrir; poderá ser um path absoluto ou relativo. A função open() retorna um objeto File.

Teste isso criando um arquivo-texto chamado *hello.txt* usando o Notepad ou o TextEdit. Digite **Hello world!** como o conteúdo desse arquivo-texto e salve-o na pasta home de seu usuário. Em seguida, se você estiver usando Windows, digite o seguinte no shell interativo:

```
>>> helloFile = open('C:\\Users\\sua_pasta_home\\hello.txt')
```

Se estiver usando o OS X, digite o seguinte no shell interativo:

```
>>> helloFile = open('/Users/sua_pasta_home/hello.txt')
```

Não se esqueça de substituir *sua_pasta_home* pelo seu nome de usuário no computador. Por exemplo, meu nome de usuário é *asweigart*, portanto devo especificar 'C:\\Users\\asweigart\\hello.txt' no Windows.

Ambos os comandos abrirão o arquivo em modo de "leitura de texto simples" – ou em *modo de leitura* (read mode) para ser mais conciso. Quando um arquivo é aberto em modo de leitura, o Python permite somente ler dados do arquivo; você não poderá escrever no arquivo nem modificá-lo de forma alguma. O modo de leitura é o modo default para os arquivos que forem abertos em Python. No entanto, se não quiser contar com os defaults do Python, você poderá especificar explicitamente o modo passando o valor de string 'r' como o segundo argumento de open(). Desse modo, open('/Users/asweigart/hello.txt', 'r') e open('/Users/asweigart/hello.txt') fazem o mesmo.

A chamada a open() retorna um objeto File. Um objeto File representa um arquivo em seu computador; é somente outro tipo de valor em Python, muito semelhante às listas e aos dicionários com os quais você já tem familiaridade. No exemplo anterior, armazenamos o objeto File na variável helloFile. Agora, sempre que quisermos ler ou escrever no arquivo, poderemos fazer isso chamando os métodos do objeto File em helloFile.

Lendo o conteúdo dos arquivos

Agora que temos um objeto File, podemos começar a ler esse arquivo. Se quiser ler todo o conteúdo de um arquivo como um valor de string, utilize o método read() do objeto File. Vamos prosseguir com o objeto File para *hello.txt* que armazenamos em helloFile . Digite o seguinte no shell interativo:

```
>>> helloContent = helloFile.read()
>>> helloContent
'Hello world!'
```

Se você pensar no conteúdo de um arquivo como um único valor de string extenso, o método read() retornará a string armazenada no arquivo.

De modo alternativo, podemos utilizar o método readlines() para obter uma *lista* de valores de string do arquivo, uma string para cada linha de texto. Por exemplo, crie um arquivo chamado *sonnet29.txt* no mesmo diretório em que está *hello.txt* e insira o texto a seguir nesse arquivo:

```
When, in disgrace with fortune and men's eyes,
I all alone beweep my outcast state,
And trouble deaf heaven with my bootless cries,
And look upon myself and curse my fate,
```

Não se esqueça de separar as quatro linhas com quebras de linha. Em seguida, digite o seguinte no shell interativo.

```
>>> sonnetFile = open('sonnet29.txt')
>>> sonnetFile.readlines()
[When, in disgrace with fortune and men's eyes,\n', ' I all alone beweep my outcast
state,\n', And trouble deaf heaven with my bootless cries,\n', And look upon myself and
curse my fate,']
```

Observe que cada um dos valores de string termina com um caractere \n de quebra de linha, exceto a última linha do arquivo. Geralmente é mais fácil trabalhar com uma lista de strings do que com um único valor de string enorme.

Escrevendo em arquivos

O Python permite escrever conteúdo em um arquivo de modo muito semelhante à maneira como a função print() "escreve" strings na tela. Contudo não podemos escrever em um arquivo aberto em modo de leitura. Em vez disso, é necessário abri-lo em modo de "escrita de texto simples" ou de "adição de texto simples", ou seja, em *modo de escrita* (write mode) e em *modo de adição* (append mode), para sermos mais concisos.

O modo de escrita sobrescreverá o arquivo existente e começará do zero, como ocorre quando o valor de uma variável é sobrescrito com um novo valor. Passe 'w' como segundo argumento de open() para abrir o arquivo em modo de escrita. O modo de adição, por outro lado, adicionará o texto no final do arquivo existente. Podemos pensar nisso como a adição em uma lista que está em uma variável em vez de sobrescrever totalmente a variável. Passe

'a' como segundo argumento de open() para abrir o arquivo em modo de adição.

Se o nome do arquivo passado para open() não existir, tanto o modo de escrita quanto o modo de adição criarão um novo arquivo vazio. Após ler ou escrever em um arquivo, chame o método close() antes de abrir o arquivo novamente.

Vamos reunir todos esses conceitos. Digite o seguinte no shell interativo:

```
>>> baconFile = open('bacon.txt', 'w')
>>> baconFile.write('Hello world!\n')
13
>>> baconFile.close()
>>> baconFile = open('bacon.txt', 'a')
>>> baconFile.write('Bacon is not a vegetable.')
25
>>> baconFile.close()
>>> baconFile = open('bacon.txt')
>>> content = baconFile.read()
>>> baconFile.close()
>>> print(content)
Hello world!
Bacon is not a vegetable.
```

Inicialmente, abrimos *bacon.txt* em modo de escrita. Como ainda não há nenhum *bacon.txt*, o Python criará um. Chamar write() no arquivo aberto e passar o argumento de string 'Hello world! /n' a essa função fará a string ser escrita no arquivo e retornará o número de caracteres escritos, incluindo o caractere de quebra de linha. Em seguida, fechamos o arquivo.

Para adicionar texto ao conteúdo existente no arquivo em vez de substituir a string que acabamos de escrever, abrimos o arquivo em modo de adição. Escrevemos 'Bacon is not a vegetable.' no arquivo e o fechamos. Por fim, para exibir o conteúdo do arquivo na tela, nós o abrimos em seu modo de leitura default, chamamos read(), armazenamos o objeto File resultante em content, fechamos o arquivo e exibimos content.

Observe que o método write() não acrescenta um caractere de quebra de linha automaticamente no final da string, como é feito pela função print(). Você deverá acrescentar esse caractere por conta própria.

Salvando variáveis com o módulo shelve

Você pode salvar variáveis em seus programas Python em arquivos shelf binários usando o módulo shelve. Dessa maneira, seu programa poderá restaurar dados em variáveis que estão no disco rígido. O módulo shelve permitirá adicionar funcionalidades Save (Salvar) e Open (Abrir) em seu programa. Por exemplo, se você executar um programa e inserir alguns parâmetros de configuração, será possível salvar essas configurações em um arquivo shelf e, em seguida, fazer o programa carregá-las na próxima vez em que for executado.

Digite o seguinte no shell interativo:

```
>>> import shelve
>>> shelfFile = shelve.open('mydata')
>>> cats = ['Zophie', 'Pooka', 'Simon']
>>> shelfFile['cats'] = cats
>>> shelfFile.close()
```

Para ler e escrever dados usando o módulo shelve, inicialmente é necessário importar esse módulo. Chame shelve.open() e passe um nome de arquivo; em seguida, armazene o valor de shelf retornado em uma variável. Você pode fazer alterações no valor de shelf como se fosse um dicionário. Quando terminar, chame close() no valor de shelf. Nesse caso, nosso valor de shelf está armazenado em shelfFile. Criamos uma lista cats e escrevemos shelfFile['cats'] = cats para armazenar a lista em shelfFile como um valor associado à chave 'cats' (como em um dicionário). Então chamamos close() em shelfFile.

Após executar o código anterior no Windows, você verá três novos arquivos no diretório de trabalho atual: *mydata.bak*, *mydata.dat* e *mydata.dir*. No OS X, somente um único arquivo *mydata.db* será criado.

Esses arquivos binários contêm os dados armazenados em seu shelf. O formato desses arquivos binários não é importante; você só precisa saber o que o módulo shelve faz, e não como ele faz. O módulo permite que você não se preocupe com o modo como os dados de seu programa são armazenados em um arquivo.

Seus programas poderão usar o módulo shelve para reabrir e obter posteriormente os dados desses arquivos shelf. Os valores de shelf não

precisam ser abertos em modo de leitura ou de escrita – ambas as operações serão permitidas após os valores serem abertos. Digite o seguinte no shell interativo:

```
>>> shelfFile = shelve.open('mydata')
>>> type(shelfFile)
<class 'shelve.DbfilenameShelf'>
>>> shelfFile['cats']
['Zophie', 'Pooka', 'Simon']
>>> shelfFile.close()
```

Nesse caso, abrimos o arquivo shelf para verificar se nossos dados foram armazenados corretamente. Especificar shelfFile['cats'] retorna a mesma lista que armazenamos anteriormente, portanto sabemos que a lista está armazenada corretamente, e então chamamos close().

Assim como os dicionários, os valores de shelf têm métodos keys() e values() que retornarão as chaves e os valores do shelf em formatos semelhantes a listas. Como esses métodos retornam valores semelhantes a listas, e não listas de verdade, você deve passá-los à função list() para obtê-los em forma de lista. Digite o seguinte no shell interativo:

```
>>> shelfFile = shelve.open('mydata')
>>> list(shelfFile.keys())
['cats']
>>> list(shelfFile.values())
[['Zophie', 'Pooka', 'Simon']]
>>> shelfFile.close()
```

O formato texto simples é útil para criar arquivos que serão lidos em um editor de texto como o Notepad ou o TextEdit, porém, se quiser salvar dados de seus programas Python, utilize o módulo shelve.

Salvando variáveis com a função pprint.pformat()

Lembre-se da seção “Apresentação elegante”, em que a função pprint.pprint() fazia uma apresentação elegante (“pretty print”) do conteúdo de uma lista ou de um dicionário na tela, enquanto a função pprint.pformat() retornava o mesmo texto na forma de uma string em vez de exibi-la. Essa string não só está

formatada para facilitar a leitura como também é um código Python sintaticamente correto. Suponha que você tenha um dicionário armazenado em uma variável e queira salvar essa variável e o seu conteúdo para usar futuramente. A chamada a pprint.pformat() fornecerá uma string que poderá ser gravada em um arquivo *.py*. Esse arquivo será seu próprio módulo, que poderá ser importado sempre que você quiser usar as variáveis armazenadas nele.

Por exemplo, digite o seguinte no shell interativo:

```
>>> import pprint
>>> cats = [{'name': 'Zophie', 'desc': 'chubby'}, {'name': 'Pooka', 'desc': 'fluffy'}]
>>> pprint.pformat(cats)
"[{'desc': 'chubby', 'name': 'Zophie'}, {'desc': 'fluffy', 'name': 'Pooka'}]"
>>> fileObj = open('myCats.py', 'w')
>>> fileObj.write('cats = ' + pprint.pformat(cats) + '\n')
83
>>> fileObj.close()
```

Nesse caso, importamos pprint para podermos usar pprint.pformat(). Temos uma lista de dicionários armazenada em uma variável cats. Para manter a lista em cats disponível mesmo após termos fechado o shell, utilizamos pprint.pformat() para retorná-la como uma string. Depois que tivermos os dados em cats na forma de uma string, será fácil gravar a string em um arquivo que chamaremos de *myCats.py*.

Os módulos que uma instrução import importa são apenas scripts Python. Quando a string de pprint.pformat() for salva em um arquivo *.py*, esse arquivo constituirá um módulo que poderá ser importado como qualquer outro.

Como os scripts Python em si são apenas arquivos-texto com a extensão de arquivo *.py*, seus programas Python podem até mesmo gerar outros programas Python. Esses arquivos podem ser então importados em scripts.

```
>>> import myCats
>>> myCats.cats
[{'name': 'Zophie', 'desc': 'chubby'}, {'name': 'Pooka', 'desc': 'fluffy'}]
>>> myCats.cats[0]
{'name': 'Zophie', 'desc': 'chubby'}
>>> myCats.cats[0]['name']
'Zophie'
```

A vantagem de criar um arquivo *.py* (em oposição a salvar variáveis com o módulo shelve) está no fato de o conteúdo do arquivo poder ser lido e modificado por qualquer pessoa que utilize um editor de texto simples, por ele é um arquivo-texto. Na maioria das aplicações, porém, salvar dados usando o módulo shelve será a maneira preferida de salvar variáveis em um arquivo. Somente tipos de dados básicos como inteiros, números de ponto flutuante, strings, listas e dicionários podem ser gravados em um arquivo como texto simples. Objetos File, por exemplo, não podem ser codificados como texto.

Projeto: gerando arquivos aleatórios de provas

Suponha que você seja um professor de geografia, tenha 35 alunos em sua classe e queira aplicar uma prova surpresa sobre as capitais dos estados norte-americanos. Infelizmente, sua classe tem alguns alunos desonestos, e não é possível confiar neles acreditando que não vão colar. Você gostaria de deixar a ordem das perguntas aleatória para que cada prova seja única, fazendo com que seja impossível para alguém copiar as respostas de outra pessoa. É claro que fazer isso manualmente seria uma tarefa demorada e maçante. Felizmente, você conhece um pouco de Python.

Eis o que o programa faz:

- Cria 35 provas diferentes.
- Cria 50 perguntas de múltipla escolha para cada prova em ordem aleatória.
- Fornece a resposta correta e três respostas incorretas aleatórias para cada pergunta em ordem aleatória.
- Grava as provas em 35 arquivos-texto.
- Grava os gabaritos contendo as respostas em 35 arquivos-texto.

Isso significa que o código deverá fazer o seguinte:

- Armazenar os estados e suas capitais em um dicionário.
- Chamar open(), write() e close() para os arquivos-texto contendo as provas e os gabaritos com as respostas.
- Usar random.shuffle() para deixar a ordem das perguntas e as opções de múltipla escolha aleatórias.

Passo 1: Armazenar os dados da prova em um dicionário

O primeiro passo consiste em criar um esqueleto de script e preenchê-lo com os dados da prova. Crie um arquivo chamado *randomQuizGenerator.py* e faça com que ele tenha a seguinte aparência:

```
   #! python3
   # randomQuizGenerator.py – Cria provas com perguntas e respostas em
   # ordem aleatória, juntamente com os gabaritos contendo as respostas.

u import random

   # Os dados para as provas. As chaves são os estados e os valores são as capitais.
v capitals = {'Alabama': 'Montgomery', 'Alaska': 'Juneau', 'Arizona': 'Phoenix',
   'Arkansas': 'Little Rock', 'California': 'Sacramento', 'Colorado': 'Denver',
   'Connecticut': 'Hartford', 'Delaware': 'Dover', 'Florida': 'Tallahassee', 'Georgia':
   'Atlanta', 'Hawaii': 'Honolulu', 'Idaho': 'Boise', 'Illinois': 'Springfield',
   'Indiana': 'Indianapolis', 'Iowa': 'Des Moines', 'Kansas': 'Topeka', 'Kentucky':
   'Frankfort', 'Louisiana': 'Baton Rouge', 'Maine': 'Augusta', 'Maryland': 'Annapolis',
   'Massachusetts': 'Boston', 'Michigan': 'Lansing', 'Minnesota': 'Saint Paul',
   'Mississippi': 'Jackson', 'Missouri': 'Jefferson City', 'Montana': 'Helena', 'Nebraska':
   'Lincoln', 'Nevada': 'Carson City', 'New Hampshire': 'Concord', 'New Jersey': 'Trenton',
   'New Mexico': 'Santa Fe', 'New York': 'Albany', 'North Carolina': 'Raleigh', 'North
   Dakota': 'Bismarck', 'Ohio': 'Columbus', 'Oklahoma': 'Oklahoma City', 'Oregon':
   'Salem', 'Pennsylvania': 'Harrisburg', 'Rhode Island': 'Providence', 'South Carolina':
   'Columbia', 'South Dakota': 'Pierre', 'Tennessee': 'Nashville', 'Texas': 'Austin',
   'Utah': 'Salt Lake City', 'Vermont': 'Montpelier', 'Virginia': 'Richmond', 'Washington':
   'Olympia', 'West Virginia': 'Charleston', 'Wisconsin': 'Madison', 'Wyoming': 'Cheyenne'}

   # Gera 35 arquivos contendo as provas.
w for quizNum in range(35):
      # TODO: Cria os arquivos com as provas e os gabaritos das respostas.

      # TODO: Escreve o cabeçalho da prova.

      # TODO: Embaralha a ordem dos estados.

      # TODO: Percorre todos os 50 estados em um loop, criando uma pergunta para cada
   um.
```

Como esse programa ordenará as perguntas e as respostas de forma

aleatória, será necessário importar o módulo random u para que seja possível utilizar suas funções. A variável capitals v contém um dicionário com os estados norte-americanos como chaves e suas capitais como os valores. Como você quer criar 35 provas, o código que gera o arquivo com a prova e com o gabarito das respostas (marcado com comentários TODO por enquanto) estará em um loop for que executará 35 vezes w. (Esse número pode ser alterado para gerar qualquer quantidade de arquivos com as provas.)

Passo 2: Criar o arquivo com a prova e embaralhar a ordem das perguntas

Agora é hora de começar a preencher aqueles TODOs.

O código no loop será repetido 35 vezes – uma para cada prova –, portanto você deverá se preocupar somente com uma prova de cada vez no loop. Inicialmente, o arquivo propriamente dito para a prova será criado. Ele deve ter um nome único de arquivo e deve também ter algum tipo de cabeçalho-padrão, com espaços para o aluno preencher o nome, a data e o período da classe. Em seguida, será necessário obter uma lista dos estados em ordem aleatória, que poderá ser usada posteriormente para criar as perguntas e as respostas da prova.

Adicione as linhas de código a seguir em *randomQuizGenerator.py*:

```
#! python3
# randomQuizGenerator.py – Cria provas com perguntas e respostas em
# ordem aleatória, juntamente com os gabaritos contendo as respostas.

--trecho removido--

# Gera 35 arquivos contendo as provas.
for quizNum in range(35):
    # Cria os arquivos com as provas e os gabaritos das respostas.
u   quizFile = open('capitalsquiz%s.txt' % (quizNum + 1), 'w')
v   answerKeyFile = open('capitalsquiz_answers%s.txt' % (quizNum + 1), 'w')

    # Escreve o cabeçalho da prova.
w   quizFile.write('Name:\n\nDate:\n\nPeriod:\n\n')
    quizFile.write((' ' * 20) + 'State Capitals Quiz (Form %s)' % (quizNum + 1))
```

```
    quizFile.write('\n\n')

    # Embaralha a ordem dos estados.
    states = list(capitals.keys())
x   random.shuffle(states)

    # TODO: Percorre todos os 50 estados em um loop, criando uma pergunta para cada
  um.
```

Os nomes dos arquivos para as provas serão *capitalsquiz<N>.txt*, em que *<N>* é um número único para a prova, proveniente de quizNum, que é o contador do loop for. O gabarito das respostas de *capitalsquiz<N>.txt* será armazenado em um arquivo-texto chamado *capitalsquiz_answers<N>.txt*. A cada passagem pelo loop, o placeholder %s em 'capitalsquiz%s.txt' e em 'capitalsquiz_answers%s.txt' será substituído por (quizNum + 1); sendo assim, a primeira prova e o primeiro gabarito criados serão *capitalsquiz1.txt* e *capitalsquiz_answers1.txt*. Esses arquivos serão criados por meio de chamadas à função open() em u e em v, com 'w' como segundo argumento para abri-los em modo de escrita.

As instruções write() em w criam um cabeçalho de prova para que o aluno possa preencher. Por fim, uma lista embaralhada dos estados norte-americanos é criada com a ajuda da função random.shuffle() x que reorganiza aleatoriamente os valores de qualquer lista recebida.

Passo 3: Criar as opções de resposta

Agora devemos gerar as opções de resposta para cada pergunta, que corresponderão às múltiplas escolhas de A a D. Será necessário criar outro loop for – esse servirá para gerar o conteúdo de cada uma das 50 perguntas da prova. Em seguida, haverá um terceiro loop for aninhado para gerar as opções de múltipla escolha para cada pergunta. Faça seu código ter o seguinte aspecto:

```
#! python3
# randomQuizGenerator.py – Cria provas com perguntas e respostas em
# ordem aleatória, juntamente com os gabaritos contendo as respostas.

--trecho removido--
```

```
    # Percorre todos os 50 estados em um loop, criando uma pergunta para cada um.
    for questionNum in range(50):

        # Obtém respostas corretas e incorretas.
u       correctAnswer = capitals[states[questionNum]]
v       wrongAnswers = list(capitals.values())
w       del wrongAnswers[wrongAnswers.index(correctAnswer)]
x       wrongAnswers = random.sample(wrongAnswers, 3)
y       answerOptions = wrongAnswers + [correctAnswer]
z       random.shuffle(answerOptions)

        # TODO: Grava a pergunta e as opções de resposta no arquivo de prova.

        # TODO: Grava o gabarito com as respostas em um arquivo.
```

A resposta correta é fácil de ser obtida – ela está armazenada como um valor no dicionário capitals u. Esse loop percorrerá os estados na lista states embaralhada, de states[0] a states[49], encontrará cada estado em capitals e armazenará a capital correspondente a esse estado em correctAnswer.

A lista de possíveis respostas incorretas é mais complicada. Podemos obtê-la duplicando *todos* os valores do dicionário capitals v, apagando a resposta correta w e selecionando três valores aleatórios dessa lista x. A função random.sample() facilita fazer essa seleção. Seu primeiro argumento é a lista em que você deseja fazer a seleção; o segundo argumento é a quantidade de valores que você quer selecionar. A lista completa de opções de resposta será a combinação dessas três respostas incorretas com a resposta correta y. Por fim, as respostas devem ser embaralhadas z para que a resposta correta não seja sempre a opção D.

Passo 4: Gravar conteúdo nos arquivos de prova e de respostas

Tudo que resta fazer é gravar a pergunta no arquivo de prova e a resposta no arquivo com o gabarito das respostas. Faça seu código ter o seguinte aspecto:

```
#! python3
# randomQuizGenerator.py – Cria provas com perguntas e respostas em
```

```
# ordem aleatória, juntamente com os gabaritos contendo as respostas.

--trecho removido--

    # Percorre todos os 50 estados em um loop, criando uma pergunta para cada um.
    for questionNum in range(50):
        --trecho removido--

        # Grava a pergunta e as opções de resposta no arquivo de prova.
        quizFile.write('%s. What is the capital of %s?\n' % (questionNum + 1,
            states[questionNum]))
u       for i in range(4):
v           quizFile.write('    %s. %s\n' % ('ABCD'[i], answerOptions[i]))
        quizFile.write('\n')

        # Grava o gabarito com as respostas em um arquivo.
w       answerKeyFile.write('%s. %s\n' % (questionNum + 1, 'ABCD'[
            answerOptions.index(correctAnswer)]))
    quizFile.close()
    answerKeyFile.close()
```

Um loop for que percorre os inteiros de 0 a 3 escreverá as opções de resposta da lista answerOptions u. A expressão 'ABCD'[i] em v trata a string 'ABCD' como um array e será avaliada como 'A','B', 'C' e então 'D' a cada respectiva iteração pelo loop.

Na última linha w, a expressão answerOptions.index(correctAnswer) encontrará o índice inteiro da resposta correta nas opções de resposta ordenadas aleatoriamente e 'ABCD'[answerOptions.index(correctAnswer)] será avaliada com a letra da resposta correta, que será gravada no arquivo contendo o gabarito.

Após executar o programa, seu arquivo *capitalsquiz1.txt* terá a aparência a seguir, embora, é claro, suas perguntas e as opções de resposta possam ser diferentes daquelas mostradas aqui, pois dependem do resultado de suas chamadas a random.shuffle():

```
Name:

Date:

Period:
```

```
                    State Capitals Quiz (Form 1)

 1. What is the capital of West Virginia?
    A. Hartford
    B. Santa Fe
    C. Harrisburg
    D. Charleston

 2. What is the capital of Colorado?
    A. Raleigh
    B. Harrisburg
    C. Denver
    D. Lincoln

 --trecho removido--
```

O arquivo-texto *capitalsquiz_answers1.txt* correspondente terá o seguinte aspecto:

```
 1. D
 2. C
 3. A
 4. C
 --trecho removido--
```

Projeto: Multiclipboard

Suponha que você tenha a tarefa maçante de preencher diversos formulários em uma página web ou em um software com vários campos de texto. O clipboard evita que seja necessário digitar o mesmo texto repetidamente. Porém somente um texto pode estar no clipboard a cada instante. Se você tiver diversas porções de texto diferentes que devam ser copiadas e coladas, será necessário marcar e copiar os mesmos dados repetidamente.

Podemos criar um programa Python que controle diversas porções de texto. Esse "multiclipboard" se chamará *mcb.pyw* (pois "mcb" é mais conciso para digitar do que "multiclipboard"). A extensão *.pyw* quer dizer que o Python não mostrará uma janela do Terminal quando executar esse programa. (Consulte o apêndice B para obter mais detalhes.)

O programa salvará cada porção de texto do clipboard com uma palavra-chave. Por exemplo, ao executar py mcb.pyw save spam, o conteúdo atual do clipboard será salvo com a palavra-chave *spam*. Esse texto poderá ser posteriormente carregado para o clipboard novamente se py mcb.pyw spam for executado. Se o usuário se esquecer das palavras-chave existentes, ele poderá executar py mcb.pyw list para que uma lista de todas as palavras-chaves seja copiada para o clipboard.

Eis o que o programa faz:

- O argumento de linha de comando para a palavra-chave é verificado.
- Se o argumento for save, o conteúdo do clipboard será salvo com a palavra-chave.
- Se o argumento for list, todas as palavras-chaves serão copiadas para o clipboard.
- Caso contrário, o texto da palavra-chave será copiado para o clipboard.

Isso significa que o código deverá fazer o seguinte:

- Ler os argumentos de linha de comando em sys.argv.
- Ler e escrever no clipboard.
- Salvar e carregar um arquivo shelf.

Se estiver usando Windows, você poderá executar facilmente esse script a partir da janela Run... (Executar) criando um arquivo batch chamado *mcb.bat* com o conteúdo a seguir:

```
@pyw.exe C:\Python34\mcb.pyw %*
```

Passo 1: Comentários e configuração do shelf

Vamos começar criando o esqueleto de um script com alguns comentários e uma configuração básica. Faça seu código ter o seguinte aspecto:

```
  #! python3
  # mcb.pyw – Salva e carrega porções de texto no clipboard.
u # Usage: py.exe mcb.pyw save <palavra-chave> – Salva clipboard na palavra-chave.
  #        py.exe mcb.pyw <palavra-chave> - Carrega palavra-chave no clipboard.
  #        py.exe mcb.pyw list – Carrega todas as palavras-chave no clipboard.

v import shelve, pyperclip, sys
```

```
w mcbShelf = shelve.open('mcb')

  # TODO: Salva conteúdo do clipboard.

  # TODO: Lista palavras-chave e carrega conteúdo.

  mcbShelf.close()
```

Colocar informações gerais de uso em comentários no início do arquivo u é uma prática comum. Se algum dia se esquecer de como seu script deve ser executado, você sempre poderá dar uma olhada nesses comentários para recordar. Em seguida, importe seus módulos v. Copiar e colar exigirá o módulo pyperclip, enquanto ler os argumentos de linha de comando exigirá o módulo sys. O módulo shelve também será útil: sempre que o usuário quiser salvar uma nova porção de texto do clipboard, você poderá salvá-la em um arquivo shelf. Então, quando o usuário quiser copiar o texto de volta ao clipboard, você abrirá o arquivo shelf e carregará o texto de volta em seu programa. O nome do arquivo shelf terá o prefixo *mcb* w.

Passo 2: Salvar o conteúdo do clipboard com uma palavra-chave

O programa realiza tarefas diferentes conforme o usuário queira salvar texto em uma palavra-chave, carregar texto no clipboard ou listar todas as palavras-chave existentes. Vamos tratar o primeiro caso. Faça seu código ter o seguinte aspecto:

```
  #! python3
  # mcb.pyw – Salva e carrega porções de texto no clipboard.
  --trecho removido--

  # Salva conteúdo do clipboard.
u if len(sys.argv) == 3 and sys.argv[1].lower() == 'save':
v         mcbShelf[sys.argv[2]] = pyperclip.paste()
  elif len(sys.argv) == 2:
w     # TODO: Lista palavras-chave e carrega conteúdo.
  mcbShelf.close()
```

Se o primeiro argumento da linha de comando (que sempre estará no índice 1 da lista sys.argv) for 'save' u, o segundo argumento da linha de comando será a palavra-chave para o conteúdo atual do clipboard. A palavra-chave será usada como chave para mcbShelf e o valor será o texto que está no clipboard no momento v.

Se houver apenas um argumento na linha de comando, suporemos que será 'list' ou uma palavra-chave para carregar o conteúdo no clipboard. Esse código será implementado mais adiante. Por enquanto, basta colocar um comentário TODO nesse ponto w.

Passo 3: Listar palavras-chaves e carregar o conteúdo de uma palavra-chave

Por fim, vamos implementar os dois casos restantes: o usuário quer carregar texto no clipboard a partir de uma palavra-chave ou quer ver uma lista de todas as palavras-chaves disponíveis. Faça seu código ter o seguinte aspecto:

```
   #! python3
   # mcb.pyw – Salva e carrega porções de texto no clipboard.
   --trecho removido--

   # Salva conteúdo do clipboard.
   if len(sys.argv) == 3 and sys.argv[1].lower() == 'save':
           mcbShelf[sys.argv[2]] = pyperclip.paste()
   elif len(sys.argv) == 2:
       # Lista palavras-chave e carrega conteúdo.
u      if sys.argv[1].lower() == 'list':
v          pyperclip.copy(str(list(mcbShelf.keys())))
       elif sys.argv[1] in mcbShelf:
w          pyperclip.copy(mcbShelf[sys.argv[1]])
   mcbShelf.close()
```

Se houver somente um argumento de linha de comando, inicialmente vamos verificar se é 'list' u. Em caso afirmativo, uma representação em string da lista de chaves do shelf será copiada para o clipboard v. O usuário poderá colar essa lista em um editor de texto que estiver aberto e lê-la.

Caso contrário, podemos supor que o argumento de linha de comando é uma palavra-chave. Se essa palavra-chave existir no shelf mcbShelf como

chave, poderemos carregar o valor no clipboard w.

É isso! Iniciar esse programa envolve passos diferentes conforme o sistema operacional utilizado pelo seu computador. Consulte o apêndice B para ver os detalhes em seu sistema operacional.

Lembre-se do programa de repositório de senhas criado no capítulo 6 que armazenava as senhas em um dicionário. Atualizar as senhas exigia alterar o código-fonte do programa. Isso não é ideal, pois os usuários comuns não se sentem à vontade alterando um código-fonte para atualizar seus softwares. Além do mais, sempre que modificar o código-fonte de um programa, você correrá o risco de introduzir novos bugs acidentalmente. Ao armazenar os dados de um programa em um local diferente do lugar em que está o código, você poderá deixar seus programas mais fáceis para outras pessoas usarem e mais resistentes a bugs.

Resumo

Os arquivos estão organizados em pastas (também chamadas de diretórios) e um path descreve a localização de um arquivo. Todo programa que estiver executando em seu computador tem um diretório de trabalho atual, que permite especificar os paths de arquivo em relação à localização atual em vez de sempre exigir a digitação do path completo (ou absoluto). O módulo os.path contém muitas funções para manipular paths de arquivo.

Seus programas também poderão interagir diretamente com o conteúdo de arquivos-texto. A função open() pode abrir esses arquivos para ler seus conteúdos na forma de uma string longa (com o método read()) ou como uma lista de strings (com o método readlines()). A função open() pode abrir arquivos em modo de escrita ou de adição para criar novos arquivos-texto ou para adicionar dados em um arquivo-texto existente, respectivamente.

Nos capítulos anteriores, usamos o clipboard como uma maneira de inserir grandes quantidades de texto em um programa em vez de digitar tudo. Agora você pode fazer seus programas lerem arquivos diretamente do disco rígido, o que é uma melhoria significativa, pois os arquivos são muito menos voláteis que o clipboard.

No próximo capítulo, aprenderemos a lidar com os arquivos propriamente

ditos, copiando, apagando, renomeando, movendo e realizando outras operações com esses arquivos.

Exercícios práticos

1. Um path relativo é relativo a quê?
2. Um path absoluto começa com qual informação?
3. O que as funções os.getcwd() e os.chdir() fazem?
4. O que são as pastas . e .. ?
5. Em *C:\bacon\eggs\spam.txt*, qual parte corresponde ao nome do diretório e qual parte é o nome base?
6. Quais são os três argumentos de "modo" que podem ser passados para a função open()?
7. O que acontecerá se um arquivo existente for aberto em modo de escrita?

8 Qual é a diferença entre os métodos read() e readlines()?

9. Que estrutura de dados um valor de shelf lembra?

Projetos práticos

Para exercitar, crie o design e implemente os programas a seguir.

Estendendo o multiclipboard

Estenda o programa multiclipboard deste capítulo para que ele tenha um argumento de linha de comando delete <*palavra-chave*> que apagará uma palavra-chave do shelf. Em seguida, acrescente um argumento de linha de comando delete que apagará *todas* as palavras-chaves.

Mad Libs

Crie um programa Mad Libs que leia arquivos-texto e permita que o usuário acrescente seus próprios textos em qualquer local em que a palavra *ADJECTIVE*, *NOUN*, *ADVERB* ou *VERB* aparecer no arquivo-texto. Por exemplo, um arquivo-texto poderá ter o seguinte aspecto:

The ADJECTIVE panda walked to the NOUN and then VERB. A nearby NOUN was unaffected by these events.

O programa deve localizar essas ocorrências e pedir que o usuário as substitua.

Enter an adjective:
silly
Enter a noun:
chandelier
Enter a verb:
screamed
Enter a noun:
pickup truck

O texto a seguir deverá ser criado:

The silly panda walked to the chandelier and then screamed. A nearby pickup truck was unaffected by these events.

O resultado deverá ser exibido na tela e salvado em um novo arquivo-texto.

Pesquisa com regex

Crie um programa que abra todos os arquivos *.txt* de uma pasta e procure todas as linhas que correspondam a uma expressão regular fornecida pelo usuário. O resultado deverá ser exibido na tela.

CAPÍTULO 9

ORGANIZANDO ARQUIVOS

No capítulo anterior, aprendemos a criar e a escrever em arquivos novos em Python. Seus programas também poderão organizar arquivos preexistentes no disco rígido. Talvez você já tenha passado pela experiência de percorrer uma pasta repleta de dezenas, centenas ou até mesmo de milhares de arquivos copiando, renomeando, movendo ou compactando todos eles manualmente. Considere tarefas como estas:

- Fazer cópias de todos os arquivos PDF (e *somente* dos arquivos PDF) de todas as subpastas de uma pasta.
- Remover os zeros iniciais dos nomes de todos os arquivos em uma pasta que contém centenas de arquivos cujos nomes são *spam001.txt*, *spam002.txt*, *spam003.txt* e assim por diante.
- Compactar o conteúdo de diversas pastas em um arquivo ZIP (que poderia ser um sistema simples de backup).

Todas essas tarefas maçantes estão suplicando para serem automatizadas em Python. Ao programar seu computador para fazer essas tarefas, podemos transformá-lo em um arquivista ágil, que jamais comete erros.

Quando começar a trabalhar com arquivos, talvez você ache útil ser capaz de ver rapidamente a extensão (*.txt*, *.pdf*, *.jpg* e assim por diante) de um arquivo. No OS X e no Linux, é bem provável que seu browser de arquivos mostre as extensões automaticamente. No Windows, as extensões de arquivo podem estar ocultas por padrão. Para mostrar as extensões, acesse Start4Control Panel4Appearance and Personalization4Folder Options (Iniciar4Painel de Controle4Aparência e Personalização4Opções de Pasta). Na aba View (Modo de exibição), em Advanced Settings (Configurações avançadas), desmarque a caixa de seleção Hide extensions for known file type (Ocultar as extensões dos tipos de arquivo conhecidos).

Módulo shutil

O módulo shutil (shell utilities, ou utilitários de shell) contém funções que permitem copiar, mover, renomear e apagar arquivos em seus programas Python. Para usar as funções de shutil, inicialmente você deverá usar import shutil.

Copiando arquivos e pastas

O módulo shutil disponibiliza funções para copiar arquivos bem como pastas inteiras.

Chamar shutil.copy(*origem*, *destino*) copiará o arquivo no path *origem* para a pasta no path *destino*. (Tanto *origem* quanto *destino* são strings.) Se *destino* for o nome de um arquivo, ele será usado como o novo nome do arquivo copiado. Essa função retorna uma string com o path do arquivo copiado.

Digite o seguinte no shell interativo para ver como shutil.copy() funciona:

```
  >>> import shutil, os
  >>> os.chdir('C:\\')
u >>> shutil.copy('C:\\spam.txt', 'C:\\delicious')
  'C:\\delicious\\spam.txt'
v >>> shutil.copy('eggs.txt', 'C:\\delicious\\eggs2.txt')
  'C:\\delicious\\eggs2.txt'
```

A primeira chamada a shutil.copy() copia o arquivo em *C:\spam.txt* para a pasta *C:\delicious*. O valor de retorno é o path do arquivo que acabou de ser copiado. Observe que, como uma pasta foi especificada como destino u, o nome original do arquivo, ou seja, *spam.txt*, foi usado para o nome do novo arquivo copiado. A segunda chamada a shutil.copy() v também copia o arquivo em *C:\eggs.txt* para a pasta *C:\delicious*, porém dá o nome *eggs2.txt* ao arquivo copiado.

Enquanto shutil.copy() copia um único arquivo, shutil.copytree() copiará uma pasta completa, com todas as pastas e os arquivos contidos. Chamar shutil.copytree(*origem*, *destino*) copiará a pasta no path *origem*, juntamente com todos os seus arquivos e suas subpastas, para a pasta no path *destino*. Os parâmetros *origem* e *destino* são strings. A função retorna uma string com o path da pasta copiada.

Digite o seguinte no shell interativo:

```
>>> import shutil, os
>>> os.chdir('C:\\')
>>> shutil.copytree('C:\\bacon', 'C:\\bacon_backup')
'C:\\bacon_backup'
```

A chamada a shutil.copytree() cria uma nova pasta chamada *bacon_backup* com o mesmo conteúdo da pasta *bacon* original. Você acabou de fazer um backup de segurança de seu bacon muito, muito precioso.

Movendo e renomeando arquivos e pastas

Chamar shutil.move(*origem*, *destino*) moverá o arquivo ou a pasta no path *origem* para o path *destino* e retornará uma string com o path absoluto da nova localidade.

Se *destino* apontar para uma pasta, o arquivo *origem* será movido para *destino* e preservará seu nome de arquivo atual. Por exemplo, digite o seguinte no shell interativo:

```
>>> import shutil
>>> shutil.move('C:\\bacon.txt', 'C:\\eggs')
'C:\\eggs\\bacon.txt'
```

Supondo que uma pasta chamada *eggs* já exista no diretório *C:*, essa chamada a shutil.move() diz para "mover *C:\bacon.txt* para a pasta *C:\eggs*".

Se já houver um arquivo *bacon.txt* em *C:\eggs*, ele será sobrescrito. Como é fácil sobrescrever arquivos acidentalmente dessa maneira, você deve tomar alguns cuidados ao usar move().

O path *destino* também pode especificar um nome de arquivo. No exemplo a seguir, o arquivo *origem* será movido *e* renomeado.

```
>>> shutil.move('C:\\bacon.txt', 'C:\\eggs\\new_bacon.txt')
'C:\\eggs\\new_bacon.txt'
```

Essa linha quer dizer: "mova *C:\bacon.txt* para a pasta *C:\eggs* e, enquanto estiver fazendo isso, renomeie esse arquivo *bacon.txt* para *new_bacon.txt*".

Ambos os exemplos anteriores funcionaram pressupondo que havia uma pasta *eggs* no diretório *C:*. Porém, se não houver nenhuma pasta *eggs*, move()

renomeará *bacon.txt* para um arquivo chamado *eggs*.

```
>>> shutil.move('C:\\bacon.txt', 'C:\\eggs')
'C:\\eggs'
```

Nesse caso, move() não pôde encontrar uma pasta chamada *eggs* no diretório *C:*; desse modo, supôs que *destino* deve especificar um nome de arquivo, e não uma pasta. Portanto o arquivo-texto *bacon.txt* foi renomeado para *eggs* (um arquivo-texto sem a extensão de arquivo *.txt*) – provavelmente, não era isso que você queria! Esse pode ser um bug difícil de ser identificado em seus programas, pois a chamada a move() pode muito bem fazer algo que poderá ser bastante diferente do que você esperava. Esse é outro motivo para ter cuidado ao usar move().

Por fim, as pastas que compõem o destino já devem existir; do contrário, o Python lançará uma exceção. Digite o seguinte no shell interativo:

```
>>> shutil.move('spam.txt', 'c:\\does_not_exist\\eggs\\ham')
Traceback (most recent call last):
  File "C:\Python34\lib\shutil.py", line 521, in move
    os.rename(src, real_dst)
FileNotFoundError: [WinError 3] The system cannot find the path specified: 'spam.txt' ->
'c:\\does_not_exist\\eggs\\ham'
During handling of the above exception, another exception occurred:
Traceback (most recent call last):
  File "<pyshell#29>", line 1, in <module>
    shutil.move('spam.txt', 'c:\\does_not_exist\\eggs\\ham')
  File "C:\Python34\lib\shutil.py", line 533, in move
    copy2(src, real_dst)
  File "C:\Python34\lib\shutil.py", line 244, in copy2
    copyfile(src, dst, follow_symlinks=follow_symlinks)
  File "C:\Python34\lib\shutil.py", line 108, in copyfile
    with open(dst, 'wb') as fdst:
FileNotFoundError: [Errno 2] No such file or directory: 'c:\\does_not_exist\\eggs\\ham'
```

O Python procura *eggs* e *ham* no diretório *does_not_exist*. Ele não encontra o diretório inexistente, portanto não poderá mover *spam.txt* para o path especificado.

Apagando arquivos e pastas permanentemente

Podemos apagar um único arquivo ou uma única pasta vazia com funções do módulo os, ao passo que, para apagar uma pasta e todo o seu conteúdo, devemos usar o módulo shutil.

- Chamar os.unlink(*path*) apagará o arquivo em *path*.
- Chamar os.rmdir(*path*) apagará a pasta em *path*. Essa pasta deve estar vazia, ou seja, não deve conter nenhum arquivo ou pasta.
- Chamar shutil.rmtree(*path*) removerá a pasta em *path*; além disso, todos os arquivos e as pastas nele contidos também serão apagados.

Tome cuidado ao usar essas funções em seus programas! Geralmente, é uma boa ideia executar seu programa inicialmente com essas chamadas comentadas, adicionando chamadas a print() para mostrar os arquivos que seriam apagados. Eis um programa Python que deveria apagar arquivos com extensão *.txt*, porém com um erro de digitação (destacado em negrito) que o faz apagar arquivos *.rxt*:

```
import os
for filename in os.listdir():
    if filename.endswith('.rxt'):
        os.unlink(filename)
```

Se houvesse algum arquivo importante terminado com *.rxt*, eles teriam sido apagados acidentalmente de forma permanente. Em vez disso, você deve inicialmente executar o programa da seguinte maneira:

```
import os
for filename in os.listdir():
    if filename.endswith('.rxt'):
        #os.unlink(filename)
        print(filename)
```

Agora a chamada a os.unlink() está comentada, portanto o Python a ignorará. Em seu lugar, o nome do arquivo que seria apagado será exibido. Executar antes essa versão do programa mostrará que você acidentalmente disse ao programa para apagar arquivos *.rxt* no lugar de arquivos *.txt*.

Depois que tiver certeza de que o programa funcionará conforme esperado, apague a linha print(filename) e remova o caractere de comentário da linha os.unlink(filename). Em seguida, execute o programa de novo para realmente

apagar os arquivos.

Apagando arquivos com segurança usando o módulo send2trash

Como a função interna shutil.rmtree() do Python apaga arquivos e pastas de modo irreversível, pode ser perigoso usá-la. Uma maneira muito melhor de apagar arquivos e pastas consiste em usar o módulo de terceiros send2trash. Esse módulo pode ser instalado por meio da execução de pip install send2trash em uma janela do Terminal. (Consulte o Apêndice A para ver uma explicação mais detalhada sobre a instalação de módulos de terceiros.)

Utilizar send2trash é muito mais seguro do que usar as funções normais do Python para apagar arquivos, pois as pastas e os arquivos serão enviados para a lixeira (recycle bin) de seu computador em vez de serem permanentemente apagados. Se um bug em seu programa apagar algo que você não pretendia com send2trash, será possível recuperar esses itens da lixeira mais tarde.

Após ter instalado send2trash, digite o seguinte no shell interativo:

```
>>> import send2trash
>>> baconFile = open('bacon.txt', 'a')  # cria o arquivo
>>> baconFile.write('Bacon is not a vegetable.')
25
>>> baconFile.close()
>>> send2trash.send2trash('bacon.txt')
```

Em geral, sempre utilize a função send2trash.send2trash() para apagar arquivos e pastas. Porém, embora enviar arquivos para a lixeira permita que você os recupere depois, isso não fará o espaço em disco ser disponibilizado do mesmo modo que a remoção permanente o fará. Se quiser que seu programa disponibilize espaço em disco, utilize as funções de os e de shutil para apagar arquivos e pastas. Observe que a função send2trash() pode somente enviar arquivos para a lixeira; ela não pode extrair arquivos dali.

Percorrendo uma árvore de diretório

Suponha que você queira renomear todos os arquivos de alguma pasta, além de todos os arquivos de todas as subpastas dessa pasta. Isso quer dizer que

você deverá percorrer a árvore de diretório, alcançando cada arquivo nesse processo. Escrever um programa que faça isso pode ser complicado; felizmente, o Python disponibiliza uma função para cuidar desse processo para você.

Vamos dar uma olhada na pasta *C:\delicious* e em seu conteúdo mostrados na figura 9.1.

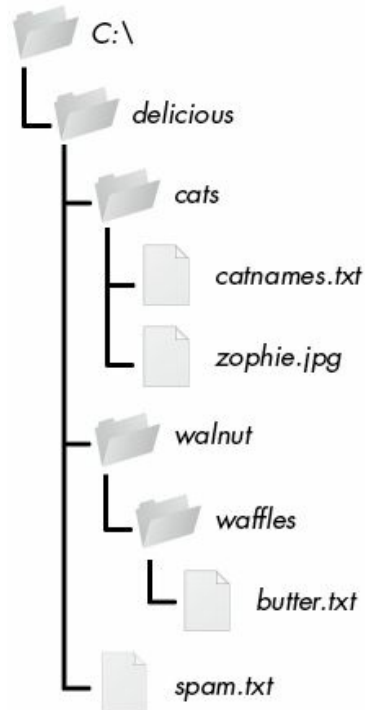



Figura 9.1 – Uma pasta de exemplo que contém três pastas e quatro arquivos.

Eis um programa de exemplo que utiliza a função os.walk() na árvore de diretório da figura 9.1:

```
import os

for folderName, subfolders, filenames in os.walk('C:\\delicious'):
    print('The current folder is ' + folderName)

    for subfolder in subfolders:
        print('SUBFOLDER OF ' + folderName + ': ' + subfolder)

    for filename in filenames:
        print('FILE INSIDE ' + folderName + ': '+ filename)
```

```
print('')
```

A função os.walk() recebe um único valor de string: o path de uma pasta. os.walk() pode ser usada em uma instrução de loop for para percorrer uma árvore de diretório, de modo muito semelhante à forma como a função range() é utilizada para percorrer um intervalo de números. De modo diferente de range(), a função os.walk() retornará três valores a cada iteração pelo loop:

1. Uma string com o nome da pasta atual.
2. Uma lista de strings com as pastas da pasta atual.
3. Uma lista de strings com os arquivos da pasta atual.

(Por pasta atual, quero dizer a pasta para a iteração atual do loop for. O diretório de trabalho atual do programa *não* é alterado por os.walk().)

Assim como podemos escolher o nome da variável i no código for i in range(10):, podemos também escolher os nomes das variáveis para os três valores listados anteriormente. Em geral, utilizo os nomes foldername, subfolders e filenames.

Ao executar esse programa, sua saída será:

```
The current folder is C:\delicious
SUBFOLDER OF C:\delicious: cats
SUBFOLDER OF C:\delicious: walnut
FILE INSIDE C:\delicious: spam.txt

The current folder is C:\delicious\cats
FILE INSIDE C:\delicious\cats: catnames.txt
FILE INSIDE C:\delicious\cats: zophie.jpg

The current folder is C:\delicious\walnut
SUBFOLDER OF C:\delicious\walnut: waffles

The current folder is C:\delicious\walnut\waffles
FILE INSIDE C:\delicious\walnut\waffles: butter.txt.
```

Como os.walk() retorna listas de strings para as variáveis subfolder e filename, essas listas podem ser usadas em seu próprios loops for. Substitua as chamadas à função print() pelo seu próprio código personalizado. (Ou, se não

precisar de um dos loops for ou de ambos, remova-os.)

Compactando arquivos com o módulo zipfile

Talvez você já tenha familiaridade com arquivos ZIP (com a extensão de arquivo *.zip*), que podem armazenar conteúdos compactados de vários outros arquivos. Compactar um arquivo reduz seu tamanho, o que é útil quando transferimos esse arquivo pela Internet. Como um arquivo ZIP também pode conter vários arquivos e subpastas, essa é uma maneira conveniente de empacotar diversos arquivos em um só. Esse arquivo único (*archive file*) pode então ser, por exemplo, anexado a um email.

Seus programas Python podem criar e abrir (ou *extrair*) arquivos ZIP usando funções do módulo zipfile. Suponha que você tenha um arquivo ZIP chamado *example.zip* cujo conteúdo está sendo mostrado na figura 9.2.

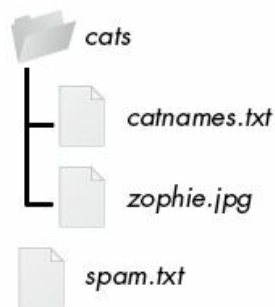



Figura 9.2 – O conteúdo de example.zip.

Você pode fazer download desse arquivo ZIP a partir de *http://nostarch.com/automatestuff/* ou simplesmente utilizar um arquivo ZIP que já esteja em seu computador.

Lendo arquivos ZIP

Para ler o conteúdo de um arquivo ZIP, inicialmente você deve criar um objeto ZipFile (observe as letras *Z* e *F* maiúsculas). Conceitualmente, os objetos ZipFile são semelhantes aos objetos File retornados pela função open() que vimos no capítulo anterior: são valores por meio dos quais o programa interage com o arquivo. Para criar um objeto ZipFile, chame a função zipfile.ZipFile() passando-lhe uma string com o nome do arquivo *.zip*. Observe que zipfile é o nome do módulo Python e ZipFile() é o nome da função.

Por exemplo, digite o seguinte no shell interativo:

```
>>> import zipfile, os
>>> os.chdir('C:\\')    # vai para a pasta que contém example.zip
>>> exampleZip = zipfile.ZipFile('example.zip')
>>> exampleZip.namelist()
['spam.txt', 'cats/', 'cats/catnames.txt', 'cats/zophie.jpg']
>>> spamInfo = exampleZip.getinfo('spam.txt')
>>> spamInfo.file_size
13908
>>> spamInfo.compress_size
3828
u >>> 'Compressed file is %sx smaller!' % (round(spamInfo.file_size /
spamInfo.compress_size, 2))
'Compressed file is 3.63x smaller!'
>>> exampleZip.close()
```

Um objeto ZipFile tem um método namelist() que retorna uma lista de strings com todos os arquivos e pastas contidos no arquivo ZIP. Essas strings podem ser passadas para o método getinfo() de ZipFile para retornar um objeto ZipInfo associado a esse arquivo em particular. Os objetos ZipInfo têm seus próprios atributos, como file_size e compress_size em bytes, que armazenam inteiros relativos ao tamanho original do arquivo e ao tamanho do arquivo compactado, respectivamente. Enquanto um objeto ZipFile representa um arquivo compactado completo, um objeto ZipInfo armazena informações úteis sobre um *único arquivo* do arquivo compactado.

O comando em u calcula a eficiência com que *example.zip* é compactado ao dividir o tamanho original do arquivo pelo tamanho do arquivo compactado e exibe essa informação usando uma string formatada com %s.

Extraindo itens de arquivos ZIP

O método extractall() de objetos ZipFile extrai todos os arquivos e as pastas de um arquivo ZIP no diretório de trabalho atual.

```
>>> import zipfile, os
>>> os.chdir('C:\\')    # vai para a pasta que contém example.zip
>>> exampleZip = zipfile.ZipFile('example.zip')
u >>> exampleZip.extractall()
```

```
>>> exampleZip.close()
```

Após executar esse código, o conteúdo de *example.zip* será extraído para *C:*. Opcionalmente, um nome de pasta pode ser passado para extractall() para que esse método extraia os arquivos em uma pasta que não seja o diretório de trabalho atual. Se a pasta passada para o método extractall() não existir, ela será criada. Por exemplo, se a chamada em u for substituída por exampleZip.extractall('C:\\delicious'), o código extrairá os arquivos de *example.zip* em uma nova pasta *C:\delicious* que será criada.

O método extract() dos objetos ZipFile extrai um único arquivo do arquivo ZIP. Prossiga com o exemplo no shell interativo:

```
>>> exampleZip.extract('spam.txt')
'C:\\spam.txt'
>>> exampleZip.extract('spam.txt', 'C:\\some\\new\\folders')
'C:\\some\\new\\folders\\spam.txt'
>>> exampleZip.close()
```

A string passada para extract() deve coincidir com uma das strings da lista retornada por namelist(). Opcionalmente, um segundo argumento pode ser passado a extract() para extrair o arquivo em uma pasta que não seja o diretório de trabalho atual. Se esse segundo argumento for uma pasta que ainda não exista, o Python a criará. O valor retornado por extract() é o path absoluto em que o arquivo foi extraído.

Criando arquivos ZIP e adicionando itens

Para criar seus próprios arquivos ZIP compactados, abra o objeto ZipFile em *modo de escrita* passando 'w' como segundo argumento. (Isso é semelhante a abrir um arquivo-texto em modo de escrita passando 'w' à função open().)

Ao passar um path para o método write() de um objeto ZipFile, o Python compactará o arquivo nesse path e o adicionará ao arquivo ZIP. O primeiro argumento do método write() é uma string que contém o nome do arquivo a ser adicionado. O segundo argumento é o parâmetro referente ao *tipo de compactação*, que diz ao computador qual algoritmo deverá ser usado para compactar os arquivos; você poderá simplesmente definir esse valor sempre com zipfile.ZIP_DEFLATED. (Isso especifica o algoritmo de compressão

deflate, que funciona bem para todos os tipos de dados.) Digite o seguinte no shell interativo:

```
>>> import zipfile
>>> newZip = zipfile.ZipFile('new.zip', 'w')
>>> newZip.write('spam.txt', compress_type=zipfile.ZIP_DEFLATED)
>>> newZip.close()
```

Esse código cria um novo arquivo ZIP chamado *new.zip* que contém o conteúdo compactado de *spam.txt*.

Tenha em mente que, assim como na escrita em arquivos, o modo de escrita apagará qualquer conteúdo existente em um arquivo ZIP. Se quiser simplesmente adicionar arquivos em um arquivo ZIP existente, passe 'a' como o segundo argumento de zipfile.ZipFile() para que o arquivo ZIP seja aberto em *modo de adição*.

Projeto: Renomeando arquivos com datas em estilo americano para datas em estilo europeu

Suponha que seu chefe envie milhares de arquivos a você por email com datas em estilo americano (MM-DD-AAAA) nos nomes dos arquivos e precise que eles sejam renomeados com datas em estilo europeu (DD-MM-AAAA). Essa tarefa maçante poderia exigir um dia inteiro para ser feita manualmente! Vamos criar um programa para fazer isso.

Eis o que o programa deve fazer:

- Procurar todos os nomes de arquivo no diretório de trabalho atual em busca de datas em estilo americano.
- Quando um arquivo for encontrado, ele deverá ser renomeado com o mês e o dia trocados para deixar a data em estilo europeu.

Isso significa que o código deverá fazer o seguinte:

- Criar uma regex que possa identificar o padrão de texto para datas em estilo americano.
- Chamar os.listdir() para encontrar todos os arquivos no diretório de trabalho.
- Percorrer todos os nomes de arquivo em um loop usando a regex para verificar se ele contém uma data.

• Se houver uma data, o arquivo deverá ser renomeado com shutil.move().

Para esse projeto, abra uma nova janela no editor de arquivo e salve seu código como *renameDates.py*.

Passo 1: Criar uma regex para datas em estilo americano

A primeira parte do programa deverá importar os módulos necessários e criar uma regex que possa identificar datas em formato MM-DD-AAAA. Os comentários TODO lembrarão você do que ainda resta ser implementado nesse programa. Digitá-los como TODO facilita localizá-los usando o recurso de pesquisa com CTRL-F do IDLE. Faça seu código ter o seguinte aspecto:

```
  #! python3
  # renameDates.py – Renomeia os nomes de arquivo com formato de data MM-DD-
  AAAA em estilo
  # americano para o formato DD-MM-AAAA em estilo europeu.

u import shutil, os, re

  # Cria uma regex que corresponda aos arquivos com formato de data em estilo
  americano.
v datePattern = re.compile(r"""^(.*?) # todo o texto antes da data
     ((0|1)?\d)-                    # um ou dois dígitos para o mês
     ((0|1|2|3)?\d)-                # um ou dois dígitos para o dia
     ((19|20)\d\d)                  # quatro dígitos para o ano
     (.*?)$                         # todo o texto após a data
w    """, re.VERBOSE)

  # TODO: Percorre os arquivos do diretório de trabalho com um loop.

  # TODO: Ignora os arquivos que não tenham uma data.

  # TODO: Obtém as diferentes partes do nome do arquivo.

  # TODO: Compõe o nome do arquivo em estilo europeu.

  # TODO: Obtém os paths absolutos completos dos arquivos.

  # TODO: Renomeia os arquivos.
```

Conforme vimos neste capítulo, sabemos que a função shutil.move() pode ser usada para renomear arquivos: seus argumentos são o nome do arquivo a ser renomeado e o nome do novo arquivo. Como essa função está no módulo shutil, ele deverá ser importado u.

Contudo, antes de renomear os arquivos, será necessário identificar quais arquivos deverão ser renomeados. Os nomes de arquivo com datas como *spam4-4-1984.txt* e *01-03-2014eggs.zip* devem ser renomeados, enquanto nomes de arquivo sem datas, por exemplo, *littlebrother.epub*, poderão ser ignorados.

Uma expressão regular poderá ser usada para identificar esse padrão. Após importar o módulo re no início, chame re.compile() para criar um objeto Regex v. Passar re.VERBOSE como segundo argumento w permitirá o uso de espaços em branco e de comentários na string de regex para torná-la mais legível.

A string da expressão regular começa com ^(.*?) para corresponder a qualquer texto no início do nome do arquivo que possa estar antes da data. O grupo ((0|1)?\d) corresponde ao mês. O primeiro dígito pode ser 0 ou 1, portanto a regex faz a correspondência de 12 para dezembro, mas também de 02 para fevereiro. Esse dígito é opcional, de modo que o mês pode ser 04 ou 4 para abril. O grupo para o dia é ((0|1|2|3)?\d) e segue uma lógica semelhante; 3, 03 e 31 são números válidos para os dias. (Sim, essa regex aceitará algumas datas inválidas como 4-31-2014, 2-29-2013 e 0-15-2014. As datas têm muitos casos especiais complicados que são facilmente esquecidos. Porém, por questões de simplicidade, a regex desse programa será suficiente.)

Embora 1885 seja um ano válido, podemos procurar somente os anos nos séculos XX ou XXI. Isso evitará que seu programa faça acidentalmente a correspondência de nomes de arquivo que não tenham datas, mas cujos formatos pareçam incluí-la, como *10-10-1000.txt*.

A parte referente a (.*?)$ da regex corresponderá a qualquer texto que estiver após a data.

Passo 2: Identificar as partes da data nos nomes de arquivo

Em seguida, o programa deverá percorrer a lista de strings contendo os

nomes dos arquivos retornada por os.listdir() em um loop e fazer a correspondência com a regex. Qualquer arquivo que não tenha uma data deverá ser ignorado. Para os nomes de arquivo que contenham uma data, o texto correspondente será armazenado em diversas variáveis. Preencha os três primeiros TODOs de seu programa com o código a seguir:

```
  #! python3
  # renameDates.py – Renomeia os nomes de arquivo com formato de data MM-DD-
  AAAA em estilo
  # americano para o formato DD-MM-AAAA em estilo europeu.
  --trecho removido--
  # Percorre os arquivos do diretório de trabalho com um loop.
  for amerFilename in os.listdir('.'):
     mo = datePattern.search(amerFilename)

     # Ignora os arquivos que não tenham uma data.
u    if mo == None:
v       continue

w    # Obtém as diferentes partes do nome do arquivo.
     beforePart = mo.group(1)
     monthPart  = mo.group(2)
     dayPart    = mo.group(4)
     yearPart   = mo.group(6)
     afterPart  = mo.group(8)
  --trecho removido--
```

Se o objeto Match retornado pelo método search() for None u, o nome do arquivo em amerFilename não corresponderá à expressão regular. A instrução continue v fará o restante do loop ser ignorado e prosseguirá para o próximo nome de arquivo.

Caso contrário, as várias strings correspondentes aos grupos da expressão regular serão armazenadas em variáveis de nomes beforePart, monthPart, dayPart, yearPart e afterPart w. As strings nessas variáveis serão utilizadas para compor o nome do arquivo em estilo europeu no próximo passo.

Para manter o controle sobre os números dos grupos, procure ler a regex do início e contabilize cada vez que encontrar um parêntese de abertura. Sem pensar no código, simplesmente crie um esquema para a expressão regular.

Isso poderá ser útil para visualizar os grupos. Por exemplo:

```
datePattern = re.compile(r"""^(1) # todo o texto antes da data
    (2 (3) )-                # um ou dois dígitos para o mês
    (4 (5) )-                # um ou dois dígitos para o dia
    (6 (7) )                 # quatro dígitos para o ano
    (8)$                     # todo o texto após a data
    """, re.VERBOSE)
```

Nesse caso, os números de **1** a **8** representam os grupos na expressão regular criada. Criar um esquema da expressão regular somente com os parênteses e os números dos grupos poderá proporcionar um entendimento mais claro de sua regex antes de prosseguir para o restante do programa.

Passo 3: Compor o novo nome de arquivo e renomear os arquivos

Como último passo, concatene as strings nas variáveis criadas no passo anterior para criar um estilo europeu de data: o dia vem antes do mês. Preencha os três TODOs restantes de seu programa com o código a seguir:

```
#! python3
# renameDates.py – Renomeia os nomes de arquivo com formato de data MM-DD-
AAAA em estilo
# americano para o formato DD-MM-AAAA em estilo europeu.
--trecho removido--
    # Compõe o nome do arquivo em estilo europeu.
u    euroFilename = beforePart + dayPart + '-' + monthPart + '-' + yearPart +
afterPart

    # Obtém os paths absolutos completos dos arquivos.
    absWorkingDir = os.path.abspath('.')
    amerFilename = os.path.join(absWorkingDir, amerFilename)
    euroFilename = os.path.join(absWorkingDir, euroFilename)

    # Renomeia os arquivos.
v    print('Renaming "%s" to "%s"...' % (amerFilename, euroFilename))
w    #shutil.move(amerFilename, euroFilename)   # remova o caractere de comentário
após
                               # os testes
```

Armazene a string concatenada em uma variável chamada euroFilename u. Em seguida, passe o nome original do arquivo em amerFilename e a nova variável euroFilename para a função shutil.move() para renomear o arquivo w.

Esse programa apresenta a chamada a shutil.move() comentada e exibe os nomes dos arquivo que serão renomeados em seu lugar v. Executar inicialmente o programa dessa maneira permitirá verificar se os arquivos serão renomeados corretamente. Então você poderá remover o caractere de comentário da chamada a shutil.move() e executar o programa de novo para renomear os arquivos.

Ideias para programas semelhantes

Há vários outros motivos pelos quais você poderá querer renomear um grande número de arquivos.

- Adicionar um prefixo no início do nome do arquivo, por exemplo, acrescentar *spam_* de modo a renomear *eggs.txt* para *spam_eggs.txt*.
- Alterar nomes de arquivo com datas em estilo europeu para datas em estilo americano.
- Remover zeros de arquivos como *spam0042.txt*.

Projeto: Fazer backup de uma pasta usando um arquivo ZIP

Suponha que você esteja trabalhando em um projeto cujos arquivos são mantidos em uma pasta chamada *C:\AlsPythonBook*. Você está preocupado em perder seu trabalho, portanto gostaria de criar arquivos ZIP que sejam “snapshots” (imagens instantâneas) de toda a pasta. Você gostaria de manter diferentes versões, portanto quer que o nome dos arquivos ZIP seja incrementado a cada vez que for criado; por exemplo, *AlsPythonBook_1.zip*, *AlsPythonBook_2.zip*, *AlsPythonBook_3.zip* e assim por diante. Isso poderia ser feito manualmente, porém é uma tarefa irritante e você poderia acidentalmente usar um número incorreto nos nomes dos arquivos ZIP. Será muito mais simples executar um programa que realize essa tarefa maçante para você.

Para esse projeto, abra uma nova janela no editor de arquivo e salve o programa como *backupToZip.py*.

Passo 1: Determinar o nome do arquivo ZIP

O código desse programa será colocado em uma função chamada backupToZip(). Isso facilitará copiar e colar a função em outros programas Python que precisem dessa funcionalidade. No final do programa, a função será chamada para realizar o backup. Faça o seu programa ter a seguinte aparência:

```
  #! python3
  # backupToZip.py – Copia uma pasta toda e seu conteúdo para
  # um arquivo ZIP cujo nome seja incrementado.

u import zipfile, os

  def backupToZip(folder):
      # Faz backup de todo o conteúdo de "folder" em um arquivo ZIP.

      folder = os.path.abspath(folder)   # garante que folder é um path absoluto

      # Determina o nome do arquivo que esse código deverá usar de acordo com
      # os arquivos já existentes.
v     number = 1
w     while True:
          zipFilename = os.path.basename(folder) + '_' + str(number) + '.zip'
          if not os.path.exists(zipFilename):
              break
          number = number + 1

x     # TODO: Cria o arquivo ZIP.
      # TODO: Percorre toda a árvore de diretório e compacta os arquivos de cada pasta.
      print('Done.')

  backupToZip('C:\\delicious')
```

Faça o básico antes: acrescente a linha shebang (#!), descreva o que o programa faz e importe os módulos zipfile e os u.

Defina uma função backupToZip() que aceite somente um parâmetro folder.

Esse parâmetro é uma string contendo o path da pasta cujo conteúdo nós devemos incluir no backup. A função determinará o nome do arquivo a ser usado para o arquivo ZIP que será criado; em seguida, a função criará o arquivo, percorrerá a pasta folder e acrescentará cada uma das subpastas e cada um dos arquivos ao arquivo ZIP. Escreva comentários TODO para esses passos no código-fonte para se lembrar de implementá-los posteriormente x.

A primeira parte, que consiste em nomear o arquivo ZIP, utiliza o nome base do path absoluto de folder. Se a pasta cujo backup está sendo feito for *C:\delicious*, o nome do arquivo ZIP deverá ser *delicious_N.zip*, em que *N* = 1 corresponde à primeira vez que o programa é executado, *N* = 2 corresponde à segunda vez e assim por diante.

Podemos determinar o *N* a ser usado verificando se *delicious_1.zip* já existe, em seguida, verificando se *delicious_2.zip* já existe e assim sucessivamente. Utilize uma variável chamada number para *N* v e incremente-a no loop que chama os.path.exists() para verificar se o arquivo existe w. O primeiro arquivo definido como inexistente fará o loop executar break, pois o nome do novo arquivo zip foi encontrado.

Passo 2: Criar o novo arquivo ZIP

Em seguida, vamos criar o arquivo ZIP. Faça o seu programa ter o seguinte aspecto:

```
  #! python3
  # backupToZip.py – Copia uma pasta toda e seu conteúdo para
  # um arquivo ZIP cujo nome seja incrementado.
  --trecho removido--
     while True:
        zipFilename = os.path.basename(folder) + '_' + str(number) + '.zip'
        if not os.path.exists(zipFilename):
           break
        number = number + 1

     # Cria o arquivo ZIP.
     print('Creating %s...' % (zipFilename))
u    backupZip = zipfile.ZipFile(zipFilename, 'w')
```

```
    # TODO: Percorre toda a árvore de diretório e compacta os arquivos de cada pasta.
    print('Done.')

backupToZip('C:\\delicious')
```

Agora que o nome do novo arquivo ZIP está armazenado na variável zipFilename, zipfile.ZipFile() pode ser chamado para criar o arquivo ZIP u. Não se esqueça de passar 'w' como segundo argumento para que o arquivo ZIP seja aberto em modo de escrita.

Passo 3: Percorrer a árvore de diretório e fazer adições ao arquivo ZIP

Agora devemos usar a função os.walk() para realizar a tarefa de listar todos os arquivos da pasta e de suas subpastas. Faça o seu programa ter o seguinte aspecto:

```
#! python3
# backupToZip.py – Copia uma pasta toda e seu conteúdo para
# um arquivo ZIP cujo nome seja incrementado.

--trecho removido--

    # Percorre toda a árvore de diretório e compacta os arquivos de cada pasta.
u   for foldername, subfolders, filenames in os.walk(folder):
        print('Adding files in %s...' % (foldername))
        # Acrescenta a pasta atual ao arquivo ZIP.
v       backupZip.write(foldername)

        # Acrescenta todos os arquivos dessa pasta ao arquivo ZIP.
w       for filename in filenames:
            newBase = os.path.basename(folder) + '_'
            if filename.startswith(newBase) and filename.endswith('.zip')
                continue   # não faz backup dos arquivos ZIP de backup
            backupZip.write(os.path.join(foldername, filename))
    backupZip.close()
    print('Done.')


backupToZip('C:\\delicious')
```

os.walk() pode ser usado em um loop for u, e, a cada iteração, esse método retornará o nome da pasta atual na iteração, as subpastas dessa pasta e os nomes dos arquivos nessa pasta.

No loop for, a pasta é adicionada ao arquivo ZIP v. O loop for aninhado pode percorrer todos os nomes de arquivo na lista filenames w. Cada um deles será adicionado ao arquivo ZIP, exceto os arquivos ZIP de backup criados anteriormente.

Ao executar esse programa, a saída gerada terá um aspecto semelhante a:

```
Creating delicious_1.zip...
Adding files in C:\delicious...
Adding files in C:\delicious\cats...
Adding files in C:\delicious\waffles...
Adding files in C:\delicious\walnut...
Adding files in C:\delicious\walnut\waffles...
Done.
```

Na segunda vez que for executado, o programa colocará todos os arquivos de *C:\delicious* em um arquivo ZIP chamado *delicious_2.zip* e assim por diante.

Ideias para programas semelhantes

Podemos percorrer uma árvore de diretório e adicionar arquivos a arquivos ZIP compactados em vários outros programas. Por exemplo, podemos criar programas que façam o seguinte:

- Percorrem uma árvore de diretório e compactem somente arquivos com determinadas extensões como *.txt* ou *.py* e nada além deles.
- Percorrem uma árvore de diretório e compactem todos os arquivos, exceto os arquivos *.txt* e *.py*.
- Encontrem a pasta em uma árvore de diretório que tenha o maior número de arquivos ou a pasta que utilize o máximo de espaço em disco.

Resumo

Mesmo que seja um usuário experiente de computador, é provável que você trabalhe manualmente com os arquivos usando o mouse e o teclado. Os

exploradores de arquivos modernos facilitam trabalhar com alguns arquivos. Porém, às vezes, é necessário realizar uma tarefa que poderá exigir horas usando o explorador de arquivos de seu computador.

Os módulos os e shutil oferecem funções para copiar, mover, renomear e apagar arquivos. Ao apagar arquivos, talvez você queira usar o módulo send2trash para movê-los para a lixeira em vez de apagá-los permanentemente. Quando escrever programas que lidem com arquivos, é uma boa ideia comentar o código que realmente copia/move/renomeia/apaga arquivos e adicionar uma chamada a print() em seu lugar para que você possa executar o programa e conferir exatamente o que ele fará.

Com frequência, precisaremos realizar essas operações não só em arquivos em uma pasta, mas também em todas as pastas dessa pasta, em todas as pastas dessas outras pastas e assim por diante. A função os.walk() permite percorrer essas pastas de modo que você possa se concentrar naquilo que seu programa deve fazer com os arquivos contidos nelas.

O módulo zipfile oferece uma maneira de compactar e extrair arquivos de arquivos *.zip* por meio do Python. Ao combinar isso com as funções de manipulação de arquivos de os e de shutil, o módulo zipfile facilita empacotar diversos arquivos em qualquer local de seu disco rígido. É muito mais simples fazer o upload desses arquivos *.zip* em sites ou enviá-los como anexos de emails do que fazer o mesmo para diversos arquivos separados.

Os capítulos anteriores deste livro forneceram o código-fonte para você copiar. Porém, ao escrever seus próprios programas, é provável que eles não estejam perfeitos na primeira vez. O próximo capítulo está centrado em alguns módulos Python que ajudarão a analisar e a depurar seus programas para que você possa fazê-los funcionar rapidamente da forma correta.

Exercícios práticos

1. Qual é a diferença entre shutil.copy() e shutil.copytree()?
2. Qual função é usada para renomear arquivos?
3. Qual é a diferença entre as funções para apagar arquivos nos módulos send2trash e shutil?
4. Os objetos ZipFile têm um método close() como o método close() de objetos

File. Qual método de ZipFile é equivalente ao método open() dos objetos File?

Projetos práticos

Para exercitar, escreva programas que executem as tarefas a seguir.

Cópia seletiva

Crie um programa que percorra uma árvore de diretórios e procure arquivos com determinada extensão (como *.pdf* ou *.jpg*). Copie esses arquivos do local em que estiverem para uma nova pasta.

Apagando arquivos desnecessários

Não é incomum que algumas pastas ou alguns arquivos enormes e desnecessários ocupem a maior parte do espaço de seu disco rígido. Se você estiver tentando disponibilizar espaço em seu computador, será muito mais vantajoso apagar os maiores arquivos indesejados. Porém será preciso localizá-los antes.

Crie um programa que percorra uma árvore de diretórios e procure arquivos ou pastas que sejam excepcionalmente grandes – por exemplo, aqueles que tenham um tamanho de arquivo maior que 100 MB. (Lembre-se de que, para obter o tamanho de um arquivo, os.path.getsize() do módulo os poderá ser utilizado.) Mostre esses arquivos na tela com seus paths absolutos.

Preenchendo as lacunas

Crie um programa que encontre todos os arquivos com um dado prefixo em uma única pasta, como *spam001.txt*, *spam002.txt* e assim por diante, e identifique qualquer lacuna na numeração (por exemplo, se houver um *spam001.txt* e um *spam003.txt*, mas não um *spam002.txt*). Faça o programa renomear todos os últimos arquivos para eliminar essas lacunas.

Como desafio adicional, crie outro programa que possa inserir lacunas em arquivos numerados para que um novo arquivo possa ser acrescentado.

CAPÍTULO 10

DEBUGGING

Agora que você tem conhecimento suficiente para criar programas mais complexos, talvez você comece a encontrar bugs não tão simples nesses programas. Este capítulo discutirá algumas ferramentas e técnicas para identificar a causa-raiz de bugs em seu programa e ajudará você a corrigi-los de modo mais rápido e com menos esforço.

Para parafrasear uma velha piada entre programadores, “escrever código corresponde a 90% de programação; o debugging do código representa os outros 90%”.

Seu computador fará somente o que você lhe disser para fazer; ele não lerá sua mente para fazer o que você *pretendia* que ele fizesse. Mesmo os programadores profissionais criam bugs o tempo todo, portanto não fique desanimado se o seu programa tiver um problema.

Felizmente, há algumas ferramentas e técnicas para identificar o que seu código está fazendo exatamente e em que ponto está errando. Inicialmente, veremos o logging e as asserções – dois recursos que poderão ajudar a detectar bugs com antecedência. Em geral, quanto mais cedo os bugs forem capturados, mais fácil será corrigi-los.

Em segundo lugar, veremos como utilizar o debugger (depurador). O debugger é um recurso do IDLE que permite executar um programa, uma instrução de cada vez; ele oferece a você a oportunidade de inspecionar os valores das variáveis enquanto seu código estiver executando e permite monitorar a mudança dos valores no decorrer do programa. Esse processo é muito mais lento que executar o programa em velocidade máxima, porém será conveniente para ver os valores propriamente ditos em um programa enquanto ele estiver executando em vez de deduzir quais poderiam ser esses valores a partir do código-fonte.

Gerando exceções

O Python gera uma exceção sempre que tenta executar um código inválido. No capítulo 3, vimos como tratar as exceções do Python com as instruções try e except de modo que seu programa pudesse se recuperar das exceções previstas. Porém também podemos gerar nossas próprias exceções no código. Gerar uma exceção é uma maneira de dizer "pare de executar o código dessa função e passe a execução do programa para a instrução except".

As exceções são geradas com uma instrução raise. No código, uma instrução raise é constituída das seguintes partes:

- a palavra-chave raise;
- uma chamada à função Exception();
- uma string com uma mensagem de erro conveniente passada para a função Exception().

Por exemplo, digite o seguinte no shell interativo:

```
>>> raise Exception('This is the error message.')
Traceback (most recent call last):
  File "<pyshell#191>", line 1, in <module>
    raise Exception('This is the error message.')
Exception: This is the error message.
```

Se não houver nenhuma instrução try e except para cuidar da instrução raise que gerou a exceção, o programa simplesmente falhará e exibirá a mensagem de erro da exceção.

Geralmente, é o código que chama a função, e não a função em si, que sabe como tratar uma exceção. Portanto, normalmente, você verá uma instrução raise em uma função e as instruções try e except no código que chama a função. Por exemplo, abra uma nova janela no editor de arquivo, digite o código a seguir e salve o programa como *boxPrint.py*:

```
  def boxPrint(symbol, width, height):
      if len(symbol) != 1:
u         raise Exception('Symbol must be a single character string.')
      if width <= 2:
v         raise Exception('Width must be greater than 2.')
      if height <= 2:
w         raise Exception('Height must be greater than 2.')
```

```
    print(symbol * width)
    for i in range(height - 2):
        print(symbol + (' ' * (width - 2)) + symbol)
    print(symbol * width)

for sym, w, h in (('*', 4, 4), ('O', 20, 5), ('x', 1, 3), ('ZZ', 3, 3)):
    try:
        boxPrint(sym, w, h)
x   except Exception as err:
y       print('An exception happened: ' + str(err))
```

Nesse caso, definimos uma função boxPrint() que aceita um caractere, uma largura e uma altura e utiliza o caractere para criar uma pequena figura de uma caixa com essa largura e essa altura. Essa caixa é exibida na tela.

Suponha que o caractere deva ser um caractere único e a largura e a altura devam ser maiores que 2. Acrescentamos instruções if para gerar exceções caso esses requisitos não sejam satisfeitos. Posteriormente, quando chamarmos boxPrint() com vários argumentos, nosso try/except cuidará dos argumentos inválidos.

Esse programa utiliza a forma except Exception as err da instrução except x. Se um objeto Exception for retornado de boxPrint() uvw, essa instrução except o armazenará em uma variável chamada err. O objeto Exception pode então ser convertido em uma string ao ser passado para str() de modo a gerar uma mensagem de erro mais amigável ao usuário y. Ao executar *boxPrint.py*, a saída terá o seguinte aspecto:

```
****
*  *
*  *
****
OOOOOOOOOOOOOOOOOOOO
O                  O
O                  O
O                  O
OOOOOOOOOOOOOOOOOOOO
An exception happened: Width must be greater than 2.
An exception happened: Symbol must be a single character string.
```

Ao usar as instruções try e except, podemos tratar os erros de modo mais

elegante em vez de deixar que o programa todo falhe.

Obtendo o traceback como uma string

Quando encontra um erro, o Python gera um baú de tesouros chamado *traceback* contendo informações de erro. O traceback inclui a mensagem de erro, o número da linha que provocou o erro e a sequência de chamadas de função que resultou no erro. Essa sequência de chamadas se chama *pilha de chamadas* (call stack).

Abra uma nova janela no editor de arquivo do IDLE, digite o programa a seguir e salve-o como *errorExample.py*:

```
def spam():
    bacon()

def bacon():
    raise Exception('This is the error message.')
spam()
```

Ao executar *errorExample.py*, a saída terá o seguinte aspecto:

```
Traceback (most recent call last):
  File "errorExample.py", line 7, in <module>
    spam()
  File "errorExample.py", line 2, in spam
    bacon()
  File "errorExample.py", line 5, in bacon
    raise Exception('This is the error message.')
Exception: This is the error message.
```

A partir do traceback, podemos ver que o erro ocorreu na linha 5, na função `bacon()`. Essa chamada em particular a `bacon()` originou-se na linha 2, na função `spam()`, que, por sua vez, foi chamada na linha 7. Em programas em que as funções podem ser chamadas de diversos lugares, a pilha de chamadas poderá ajudar você a determinar qual chamada conduziu ao erro.

O traceback é exibido pelo Python sempre que uma exceção gerada não é tratada. Porém também podemos obtê-lo como uma string chamando `traceback.format_exc()`. Essa função será útil se quisermos obter informações do traceback de uma exceção, mas também quisermos que uma instrução `except`

trate a exceção de modo elegante. Será necessário importar o módulo traceback do Python antes de chamar essa função.

Por exemplo, em vez de fazer o programa falhar assim que uma exceção ocorrer, nós podemos gravar as informações de traceback em um arquivo de log e manter o programa executando. Você poderá dar uma olhada no arquivo de log posteriormente, quando estiver pronto para depurar o programa. Digite o seguinte no shell interativo:

```
>>> import traceback
>>> try:
        raise Exception('This is the error message.')
except:
        errorFile = open('errorInfo.txt', 'w')
        errorFile.write(traceback.format_exc())
        errorFile.close()
        print('The traceback info was written to errorInfo.txt.')

116
The traceback info was written to errorInfo.txt.
```

116 é o valor de retorno do método write(), pois 116 caracteres foram gravados no arquivo. O texto do traceback foi gravado em *errorInfo.txt*.

```
Traceback (most recent call last):
  File "<pyshell#28>", line 2, in <module>
Exception: This is the error message.
```

Asserções

Uma *asserção* (assertion) é uma verificação de sanidade para garantir que seu código não está fazendo nada obviamente incorreto. Essas verificações de sanidade são realizadas por instruções assert. Se a verificação de sanidade falhar, uma exceção AssertionError será gerada. No código, uma instrução assert é constituída das seguintes partes:

- a palavra-chave assert;
- uma condição (ou seja, uma expressão avaliada como True ou False);
- uma vírgula;
- uma string a ser exibida quando a condição for False.

Por exemplo, digite o seguinte no shell interativo:

```
>>> podBayDoorStatus = 'open'
>>> assert podBayDoorStatus == 'open', 'The pod bay doors need to be "open".'
>>> podBayDoorStatus = 'I\'m sorry, Dave. I\'m afraid I can't do that.''
>>> assert podBayDoorStatus == 'open', 'The pod bay doors need to be "open".'
Traceback (most recent call last):
  File "<pyshell#10>", line 1, in <module>
    assert podBayDoorStatus == 'open', 'The pod bay doors need to be "open".'
AssertionError: The pod bay doors need to be "open".
```

Nesse caso, definimos podBayDoorStatus com 'open', portanto, a partir de agora, esperamos que o valor dessa variável seja sempre 'open'. Em um programa que utilize essa variável, poderemos criar muitos códigos supondo que o valor dessa variável seja 'open' – códigos que dependerão desse valor ser 'open' para que funcionem conforme esperado. Sendo assim, adicionamos uma asserção para garantir que estamos certos em supor que podBayDoorStatus seja 'open'. Nesse caso, incluímos a mensagem 'The pod bay doors need to be "open".' para que seja mais fácil ver o que há de errado se a asserção falhar.

Depois, suponha que tenhamos cometido o erro óbvio de atribuir outro valor a podBayDoorStatus, porém não percebemos isso entre tantas linhas de código. A asserção capturará esse erro e nos informará claramente o que está errado.

Falando de forma simples, uma instrução assert diz: “afirmo que esta condição é verdadeira, mas, se não for, há um bug em algum lugar do programa”. De modo diferente das exceções, seu código *não* deverá tratar as instruções assert com try e except; se um assert falhar, seu programa *deve* falhar. Ao falhar rapidamente dessa maneira, você reduzirá o tempo entre a causa original do bug e o momento em que ele será percebido pela primeira vez. Isso reduzirá a quantidade de código que deverá ser verificado até encontrar aquele que está provocando o bug.

As asserções servem para identificar erros aos programadores, e não aos usuários. Os erros dos quais pode haver uma recuperação (por exemplo, um arquivo não encontrado ou a inserção de dados inválidos pelo usuário) devem gerar uma exceção em vez de serem detectados por uma instrução assert.

Usando uma asserção em uma simulação de semáforo

Suponha que você esteja criando um programa de simulação de semáforo. A estrutura de dados que representa os semáforos em uma intersecção é um dicionário com chaves 'ns' e 'ew' para os semáforos voltados na direção norte-sul e leste-oeste, respectivamente. Os valores dessas chaves serão as strings 'green', 'yellow' ou 'red'. O código será semelhante a:

```
market_2nd = {'ns': 'green', 'ew': 'red'}
mission_16th = {'ns': 'red', 'ew': 'green'}
```

Essas duas variáveis representam as intersecções entre Market Street e 2nd Street e entre Mission Street e 16th Street. Para iniciar o projeto, queremos criar uma função switchLights() que aceitará um dicionário referente a uma intersecção como argumento e mudará as luzes do semáforo.

Inicialmente, talvez você ache que switchLights() deva simplesmente mudar cada semáforo para a próxima cor na sequência: qualquer valor 'green' deve ser alterado para 'yellow', valores 'yellow' devem mudar para 'red' e valores 'red' devem mudar para 'green'. O código para implementar essa ideia pode ter o seguinte aspecto:

```
def switchLights(stoplight):
    for key in stoplight.keys():
        if stoplight[key] == 'green':
            stoplight[key] = 'yellow'
        elif stoplight[key] == 'yellow':
            stoplight[key] = 'red'
        elif stoplight[key] == 'red':
            stoplight[key] = 'green'

switchLights(market_2nd)
```

Talvez você já tenha notado o problema com esse código, mas vamos fingir que você tenha escrito o restante do código da simulação – com milhares de linhas – sem percebê-lo. Ao executar finalmente a simulação, o programa não falhará – mas seus carros virtuais colidirão!

Como o restante do programa já foi escrito, você não tem ideia do local em que o bug poderia estar. Talvez esteja no código que simula os carros ou no

código que simula os motoristas virtuais. Você poderia gastar horas para identificar o bug na função switchLights().

Porém, se você tivesse acrescentado uma asserção enquanto escrevia a função switchLights() para verificar se *pelo menos um dos semáforos tem sempre a luz vermelha*, poderia ter incluído o seguinte no final da função:

```
assert 'red' in stoplight.values(), 'Neither light is red! ' + str(stoplight)
```

Com essa asserção definida, seu programa falharia com a seguinte mensagem de erro:

```
   Traceback (most recent call last):
     File "carSim.py", line 14, in <module>
       switchLights(market_2nd)
     File "carSim.py", line 13, in switchLights
       assert 'red' in stoplight.values(), 'Neither light is red! ' + str(stoplight)
u AssertionError: Neither light is red! {'ns': 'yellow', 'ew': 'green'}
```

A linha importante nesse caso é a que contém AssertionError u. Embora não seja ideal seu programa falhar, isso mostrará imediatamente que uma verificação de sanidade falhou: nenhuma das direções do tráfego tinha um semáforo com luz vermelha, o que significa que esse tráfego poderia estar fluindo em ambas as direções. Se houver uma falha com bastante antecedência na execução do programa, muito esforço de debugging poderá ser evitado no futuro.

Desabilitando as asserções

As asserções podem ser desabilitadas se a opção -O for passada na execução do Python. Isso será conveniente quando você tiver acabado de escrever e de testar seu programa e não quiser que ele fique lento por realizar verificações de sanidade (embora, na maior parte do tempo, as instruções assert não causem uma diferença perceptível de velocidade). As asserções devem ser usadas no desenvolvimento, e não no produto final. Quando seu programa for disponibilizado para outra pessoa executá-lo, ele deverá estar livre de bugs e não deverá exigir verificações de sanidade. Consulte o Apêndice B para obter detalhes sobre como iniciar seus programas supostamente saneados com a opção -O.

Logging

Se você já usou uma instrução print() em seu código para exibir o valor de alguma variável enquanto seu programa estava executando, então você utilizou uma forma de *logging* para depurar o seu código. O logging é uma ótima maneira de entender o que está acontecendo em seu programa e em que ordem está ocorrendo. O módulo logging do Python facilita criar um registro de mensagens personalizadas. Essas mensagens de log descreverão quando a execução do programa alcançou a chamada da função de logging e listarão qualquer variável que você tenha especificado naquele ponto. Por outro lado, uma mensagem de log ausente indica que uma parte do código foi ignorada e jamais foi executada.

Utilizando o módulo logging

Para habilitar o módulo logging e exibir as mensagens de log em sua tela à medida que seu programa executar, copie o seguinte no início de seu programa (mas depois da linha shebang #! python):

```
import logging
logging.basicConfig(level=logging.DEBUG, format=' %(asctime)s - %(levelname)s - %
(message)s')
```

Não é preciso se preocupar demais com o modo como isso funciona, mas, basicamente, quando o Python faz log de um evento, um objeto LogRecord contendo informações sobre esse evento será criado. A função basicConfig() do módulo logging permite especificar quais detalhes do objeto LogRecord você vai querer ver e como você quer que esses detalhes sejam exibidos.

Suponha que você tenha criado uma função para calcular o *fatorial* de um número. Em matemática, o fatorial de 4 é 1 × 2 × 3 × 4, ou seja, 24. O fatorial de 7 é 1 × 2 × 3 × 4 × 5 × 6 × 7, ou seja, 5.040. Abra uma nova janela no editor de arquivo e insira o código a seguir. Ele contém um bug, porém você inserirá diversas mensagens de log para ajudar a descobrir o que está errado. Salve o programa como *factorialLog.py*.

```
import logging
logging.basicConfig(level=logging.DEBUG, format=' %(asctime)s - %(levelname)s - %
(message)s')
```

```
logging.debug('Start of program')

def factorial(n):
    logging.debug('Start of factorial(%s%%)'  % (n))
    total = 1
    for i in range(n + 1):
        total *= i
        logging.debug('i is ' + str(i) + ', total is ' + str(total))
    logging.debug('End of factorial(%s%%)'  % (n))
    return total

print(factorial(5))
logging.debug('End of program')
```

Nesse caso, usamos a função logging.debug() para exibir informações de log. Essa função debug() chamará basicConfig(), e uma linha contendo informações será exibida. Essas informações estarão no formato que especificamos em basicConfig() e incluirão as mensagens passadas para debug(). A chamada a print(factorial(5)) faz parte do programa original, portanto o resultado será exibido mesmo que as mensagens de logging estejam desabilitadas.

A saída desse programa terá a seguinte aparência:

```
2015-05-23 16:20:12,664 - DEBUG - Start of program
2015-05-23 16:20:12,664 - DEBUG - Start of factorial(5)
2015-05-23 16:20:12,665 - DEBUG - i is 0, total is 0
2015-05-23 16:20:12,668 - DEBUG - i is 1, total is 0
2015-05-23 16:20:12,670 - DEBUG - i is 2, total is 0
2015-05-23 16:20:12,673 - DEBUG - i is 3, total is 0
2015-05-23 16:20:12,675 - DEBUG - i is 4, total is 0
2015-05-23 16:20:12,678 - DEBUG - i is 5, total is 0
2015-05-23 16:20:12,680 - DEBUG - End of factorial(5)
0
2015-05-23 16:20:12,684 - DEBUG - End of program
```

A função factorial() retorna 0 como o fatorial de 5, o que está incorreto. O loop for deveria multiplicar o valor em total pelos números de 1 a 5. Contudo as mensagens de log exibidas por logging.debug() mostram que a variável i está começando em 0, e não em 1. Como zero multiplicado por qualquer valor é zero, o restante das iterações também terá o valor incorreto em total. As

mensagens de logging oferecem uma trilha de migalhas de pão que ajudará a descobrir em que momento o código começou a errar.

Altere a linha for i in range(n + 1): para for i in range(**1,** n + 1): e execute o programa novamente. A saída será semelhante a:

```
2015-05-23 17:13:40,650 - DEBUG - Start of program
2015-05-23 17:13:40,651 - DEBUG - Start of factorial(5)
2015-05-23 17:13:40,651 - DEBUG - i is 1, total is 1
2015-05-23 17:13:40,654 - DEBUG - i is 2, total is 2
2015-05-23 17:13:40,656 - DEBUG - i is 3, total is 6
2015-05-23 17:13:40,659 - DEBUG - i is 4, total is 24
2015-05-23 17:13:40,661 - DEBUG - i is 5, total is 120
2015-05-23 17:13:40,661 - DEBUG - End of factorial(5)
120
2015-05-23 17:13:40,666 - DEBUG - End of program
```

A chamada a factorial(5) retorna 120 corretamente. As mensagens de log mostraram o que estava ocorrendo no loop, o que nos levou diretamente ao bug.

Podemos ver que as chamadas a logging.debug() exibiram não só as strings passadas a ela, mas também um timestamp e a palavra *DEBUG*.

Não faça debug com print()

Digitar import logging e logging.basicConfig(level=logging.DEBUG, format='%(asctime)s - %(levelname)s - %(message)s'), de certo modo, é complicado. Talvez você queira usar chamadas a print() em vez de utilizar o logging, porém não se deixe cair nessa tentação! Após terminar o debugging, você acabará gastando muito tempo removendo as chamadas a print() de seu código para cada mensagem de log. Você poderá até mesmo remover acidentalmente algumas chamadas a print() que poderiam estar sendo usadas para mensagens que não sejam de log. O interessante sobre as mensagens de log é que você tem a liberdade de encher seu programa com quantas mensagens de logs você quiser e sempre poderá desabilitá-las posteriormente ao adicionar uma única chamada a logging.disable(logging.CRITICAL). De modo diferente de print(), o módulo logging facilita alternar entre mostrar e ocultar as mensagens de log.

Essas mensagens de log se destinam ao programador, e não ao usuário. O

usuário não estará interessado no conteúdo do valor de algum dicionário que você precise ver para auxiliar no debugging; utilize uma mensagem de log para algo desse tipo. Para mensagens que o usuário queira ver, por exemplo, *File not found* (Arquivo não encontrado) ou *Invalid input, please enter a number* (Dado de entrada inválido; por favor, digite um número), use uma chamada a print(). Você não vai querer privar o usuário de ver informações úteis após ter desabilitado as mensagens de log.

Níveis de logging

Os *níveis de logging* oferecem uma maneira de classificar suas mensagens de log de acordo com a importância. Há cinco níveis de logging, descritos na tabela 10.1, do menos para o mais importante. As mensagens podem ser registradas no log com qualquer nível por meio de uma função diferente de logging.

Tabela 10.1 – Níveis de logging em Python

Nível	Função de logging	Descrição
DEBUG	logging.debug()	É o nível mais baixo. Usado para pequenos detalhes. Geralmente, você estará interessado nessas mensagens somente quando estiver diagnosticando problemas.
INFO	logging.info()	Usado para registrar informações sobre eventos em geral em seu programa ou para confirmar se tudo está funcionando nesse ponto do programa.
WARNING	logging.warning()	Usado para indicar um problema em potencial que não impede o programa de funcionar, porém poderá fazer isso no futuro.
ERROR	logging.error()	Usado para registrar um erro que fez o programa falhar em fazer algo.
CRITICAL	logging.critical()	É o nível mais alto. Usado para indicar um erro fatal que fez ou está prestes a fazer o programa parar totalmente de executar.

Sua mensagem de logging será passada como uma string a essas funções. Os níveis de logging são sugestões. Em última instância, cabe a você decidir em qual categoria sua mensagem de log se enquadra. Digite o seguinte no shell interativo:

```
>>> import logging
>>> logging.basicConfig(level=logging.DEBUG, format=' %(asctime)s -  %
(levelname)s -  %(message)s')
>>> logging.debug('Some debugging details.')
2015-05-18 19:04:26,901 - DEBUG - Some debugging details.
>>> logging.info('The logging module is working.')
2015-05-18 19:04:35,569 - INFO - The logging module is working.
>>> logging.warning('An error message is about to be logged.')
2015-05-18 19:04:56,843 - WARNING - An error message is about to be logged.
>>> logging.error('An error has occurred.')
2015-05-18 19:05:07,737 - ERROR - An error has occurred.
>>> logging.critical('The program is unable to recover!')
2015-05-18 19:05:45,794 - CRITICAL - The program is unable to recover!
```

A vantagem dos níveis de logging está no fato de você poder alterar a prioridade da mensagem de logging que você quer ver. Passar logging.DEBUG para o argumento nomeado level da função basicConfig() mostrará as mensagens de todos os níveis de logging (DEBUG é o nível mais baixo). Entretanto, após desenvolver um pouco mais o seu programa, talvez você se interesse somente pelos erros. Nesse caso, o argumento level de basicConfig() poderá ser definido para logging.ERROR. Isso fará somente as mensagens ERROR e CRITICAL serem mostradas, enquanto as mensagens DEBUG, INFO e WARNING serão ignoradas.

Desabilitando o logging

Após ter feito o debugging de seu programa, talvez você não queira todas essas mensagens de log enchendo a tela. A função logging.disable() as desabilita de modo que não seja necessário alterar o seu programa e remover todas as chamadas de logging manualmente. Basta passar um nível de logging a logging.disable() e todas as mensagens de log desse nível ou abaixo dele serão suprimidas. Sendo assim, se você quiser desabilitar totalmente o logging, basta adicionar logging.disable(logging.CRITICAL) ao seu programa. Por exemplo, digite o seguinte no shell interativo:

```
>>> import logging
>>> logging.basicConfig(level=logging.INFO, format=' %(asctime)s -  %
(levelname)s -  %(message)s')
>>> logging.critical('Critical error! Critical error!')
```

```
2015-05-22 11:10:48,054 - CRITICAL - Critical error! Critical error!
>>> logging.disable(logging.CRITICAL)
>>> logging.critical('Critical error! Critical error!')
>>> logging.error('Error! Error!')
```

Como logging.disable() desabilitará todas as mensagens após a sua chamada, provavelmente, você vai querer adicioná-la próxima à linha de código contendo import logging em seu programa. Dessa maneira, ela poderá ser facilmente encontrada para ser comentada ou não, de modo que as mensagens de logging sejam habilitadas ou desabilitadas conforme for necessário.

Logging em um arquivo

Em vez de exibir as mensagens de log na tela, podemos gravá-las em um arquivo-texto. A função logging.basicConfig() aceita um argumento nomeado filename da seguinte maneira:

```
import logging
logging.basicConfig(filename='myProgramLog.txt', level=logging.DEBUG, format='
%(asctime)s -  %(levelname)s -  %(message)s')
```

As mensagens de log serão salvadas em *myProgramLog.txt*. Apesar de serem úteis, as mensagens de logging podem congestionar sua tela e dificultar a leitura da saída do programa. Gravar mensagens de logging em um arquivo manterá sua tela limpa e fará suas mensagens serem armazenadas para que possam ser lidas após a execução do programa. Esse arquivo-texto poderá ser aberto em qualquer editor de texto, por exemplo, o Notepad ou o TextEdit.

Debugger do IDLE

O *debugger* é um recurso do IDLE que permite executar o seu programa uma linha de cada vez. O debugger executa uma única linha de código e espera você lhe dizer para continuar. Ao executar seu programa "no debugger" dessa maneira, você disporá do tempo que quiser para analisar os valores das variáveis em qualquer ponto durante o tempo de vida do programa. É uma ferramenta valiosa para identificar bugs.

Para habilitar o debugger do IDLE, clique em Debug4Debugger na janela

do shell interativo. Isso fará a janela Debug Control (Controle de debug), cuja aparência é semelhante àquela mostrada na figura 10.1, ser apresentada.

Figura 10.1 – A janela Debug Control (Controle de debug).

Quando a janela Debug Control aparecer, marque as quatro caixas de seleção Stack (Pilha), Locals (Locais), Source (Código-Fonte) e Globals (Globais) para que a janela mostre o conjunto completo de informações de debug. Enquanto a janela Debug Control estiver sendo apresentada, sempre que um programa for executado no editor de arquivo, o debugger fará uma pausa na execução antes da primeira instrução e apresentará o seguinte:

- a linha de código que está prestes a ser executada;
- uma lista de todas as variáveis locais e seus valores;
- uma lista de todas as variáveis globais e seus valores.

Você perceberá que, na lista de variáveis globais, há diversas variáveis que não foram definidas por você, como __builtins__, __doc__, __file__ e assim por diante. São variáveis que o Python define automaticamente sempre que um programa é executado. O significado dessas variáveis está além do escopo deste livro e você poderá ignorá-las sem se preocupar.

O programa fará uma pausa até que você pressione um dos cinco botões da janela Debug Control: Go, Step, Over, Out ou Quit .

Go

Clicar no botão Go fará o programa executar normalmente até terminar ou até que um *breakpoint* seja alcançado. (Os breakpoints serão descritos mais adiante neste capítulo.) Se o debugging estiver concluído e você quiser que o programa continue normalmente, clique no botão Go.

Step

Clicar no botão Step fará o debugger executar a próxima linha de código e outra pausa será feita novamente. A lista de variáveis locais e globais da janela Debug Control será atualizada caso seus valores sejam alterados. Se a próxima linha de código for uma chamada de função, o debugger "entrará" nessa função e passará para a sua primeira linha de código.

Over

Clicar no botão Over fará a próxima linha de código ser executada, de modo semelhante ao que ocorre se você clicar no botão Step. Entretanto, se a próxima linha de código for uma chamada de função, o botão Over "pulará" o código da função. O código da função será executado em velocidade máxima e o debugger fará uma pausa assim que a chamada da função retornar. Por exemplo, se a próxima linha de código for uma chamada a print(), você não estará interessado no código da função interna print(); você simplesmente quer que a string passada a ela seja exibida na tela. Por esse motivo, usar o botão Over será mais comum que utilizar o botão Step.

Out

Clicar no botão Out fará o debugger executar as linhas de código em velocidade máxima até retornar da função em que estiver no momento. Se você entrou em uma chamada de função com o botão Step e agora simplesmente quer continuar executando as instruções até retornar, clique no botão Out para "sair" da chamada de função atual.

Quit

Se quiser interromper totalmente o debugging e não se importar em continuar com a execução do restante do programa, clique no botão Quit. O botão Quit

encerrará imediatamente o programa. Se quiser executar o programa normalmente mais uma vez, selecione Debug4Debugger novamente para desabilitar o debugger.

Fazendo debugging de um programa que soma números

Abra uma nova janela no editor de arquivo e insira o código a seguir:

```
print('Enter the first number to add:')
first = input()
print('Enter the second number to add:')
second = input()
print('Enter the third number to add:')
third = input()
print('The sum is ' + first + second + third)
```

Salve esse programa como *buggyAddingProgram.py* e execute-o inicialmente sem o debugger habilitado. A saída do programa terá o seguinte aspecto:

```
Enter the first number to add:
5
Enter the second number to add:
3
Enter the third number to add:
42
The sum is 5342
```

O programa não provocou uma falha, porém a soma está obviamente incorreta. Vamos habilitar a janela Debug Control e executar o programa novamente, desta vez no debugger.

Ao pressionar F5 ou selecionar Run4Run Module (Executar4Executar módulo) – com Debug4Debugger habilitado e todas as quatro caixas de seleção marcadas na janela Debug Control) –, o programa iniciará com uma pausa na linha 1. O debugger sempre fará uma pausa na linha de código que está prestes a executar. A janela Debug Control terá uma aparência semelhante à mostrada na figura 10.2.

Figura 10.2 – A janela Debug Control quando o programa é iniciado no debugger.

Clique no botão Over uma vez para executar a primeira chamada a print(). Over deverá ser usado em vez de Step nesse caso, pois não queremos entrar no código da função print(). A janela Debug Control será atualizada para a linha 2, que estará marcada na janela do editor de arquivo, conforme mostrado na figura 10.3. Isso mostra em que ponto está a execução do programa no momento.

Clique em Over novamente para executar a chamada de função input(); os botões na janela Debug Control estarão desabilitados enquanto o IDLE espera você digitar algo para a chamada a input() na janela do shell interativo. Digite **5** e tecle Return. Os botões da janela Debug Control serão habilitados novamente.

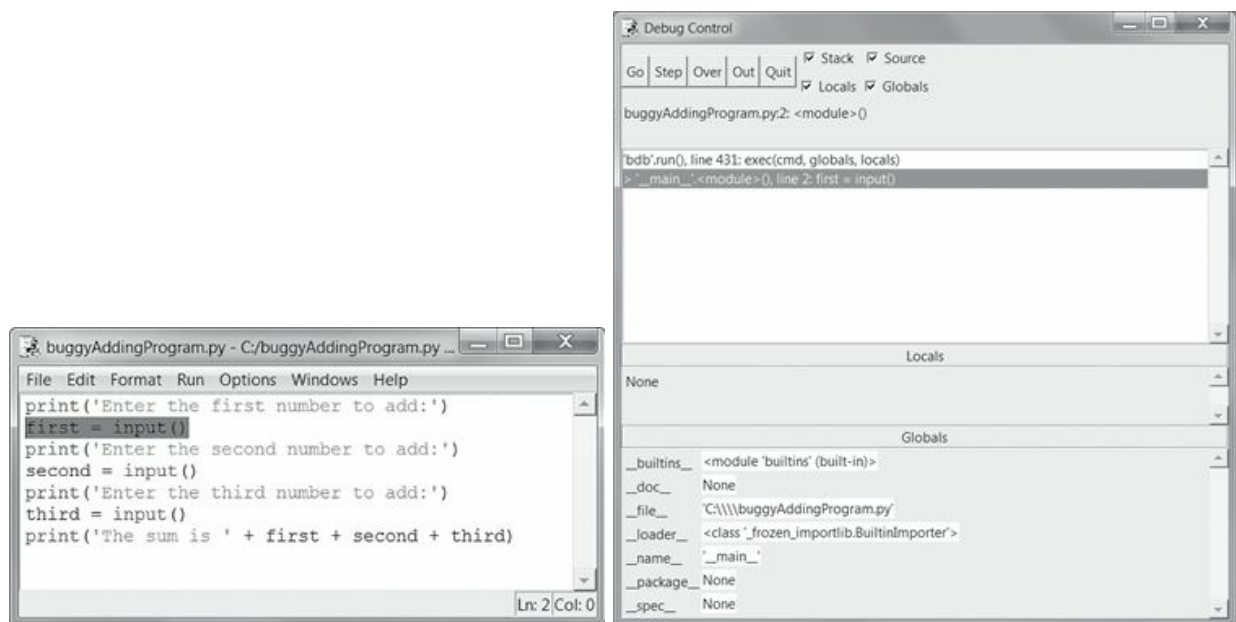



Figura 10.3 – A janela Debug Control após clicar em Over.

Continue clicando em Over e digite **3** e **42** para os dois próximos números até que o debugger esteja na linha 7, que contém a última chamada a print() do programa. A janela Debug Control deverá ter uma aparência semelhante à mostrada na figura 10.4.

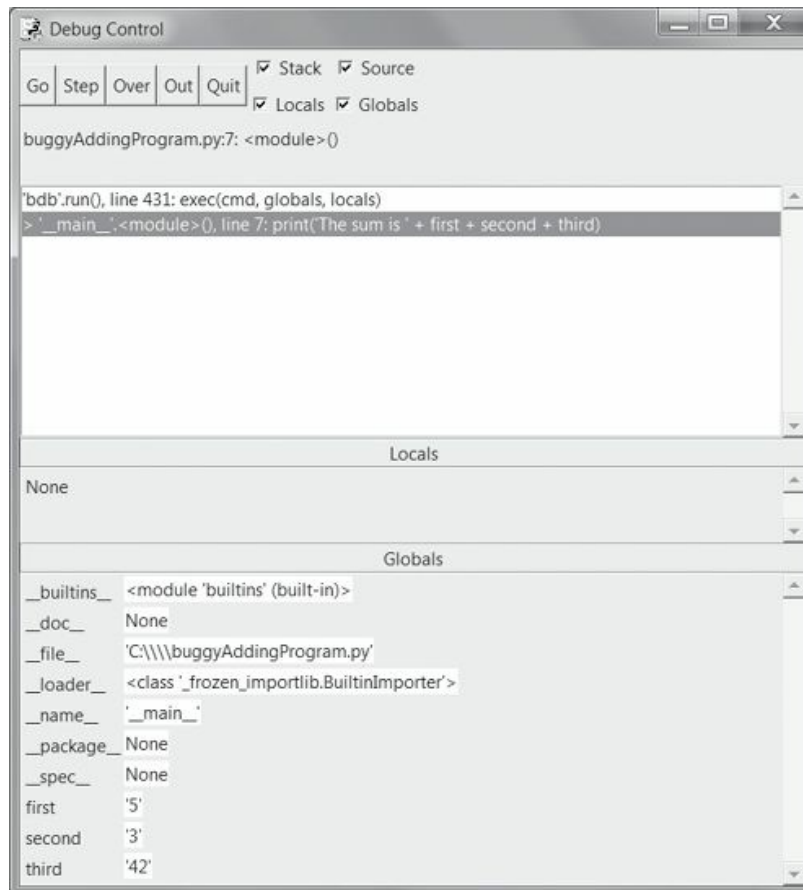



Figura 10.4 – A janela Debug Control na última linha. As variáveis estão definidas como strings, provocando o bug.

Na seção Globals, podemos ver que a primeira, a segunda e a terceira variáveis estão definidas com valores de string '5', '3' e '42', e não com os valores inteiros 5, 3 e 42. Quando a última linha é executada, essas strings são concatenadas em vez de serem somadas, provocando o bug.

Executar o programa passo a passo com o debugger é conveniente, porém poderá ser um processo lento. Geralmente você vai querer que o programa execute normalmente até que determinada linha de código seja alcançada. O debugger pode ser configurado para fazer isso por meio de breakpoints.

Breakpoints

Um *breakpoint* pode ser definido em uma linha específica de código e forçará o debugger a fazer uma pausa sempre que a execução do programa alcançar essa linha. Abra uma nova janela no editor de arquivo e digite o programa a seguir, que simula o lançamento de uma moeda mil vezes. Salve-o como

coinFlip.py.

```
   import random
   heads = 0
   for i in range(1, 1001):
u      if random.randint(0, 1) == 1:
           heads = heads + 1
       if i == 500:
v          print('Halfway done!')
   print('Heads came up ' + str(heads) + ' times.')
```

A chamada a random.randint(0, 1) u retornará 0 na metade das vezes e 1 na outra metade. Isso pode ser usado para simular um lançamento de moeda a 50/50, em que 1 representa cara. Ao ser executado sem o debugger, esse programa apresentará rapidamente uma saída como:

```
Halfway done!
Heads came up 490 times.
```

Se o programa for executado no debugger, será necessário clicar no botão Over centenas de vezes até o programa terminar. Se você estiver interessado no valor de heads na metade da execução do programa, quando 500 dos 1.000 lançamentos da moeda tiverem sido feitos, simplesmente defina um breakpoint na linha print('Halfway done!') v. Para definir um breakpoint, clique com o botão direito do mouse na linha do editor de arquivo e selecione Set Breakpoint (Definir breakpoint), conforme mostrado na figura 10.5.

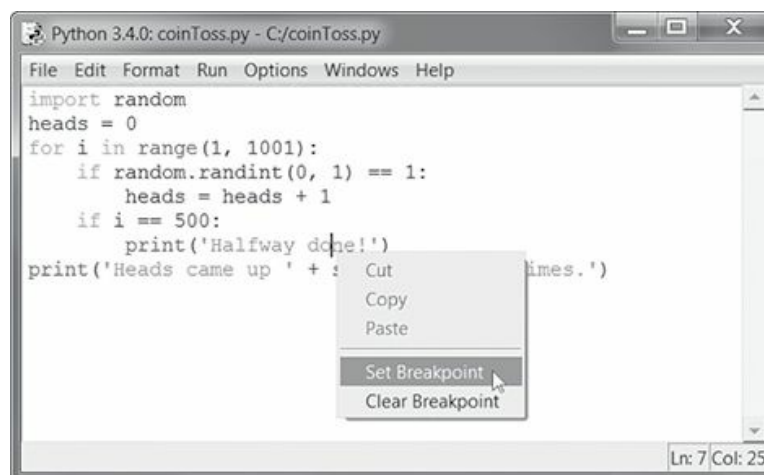



Figura 10.5 – Definindo um breakpoint.

Você não deve definir um breakpoint na linha contendo a instrução if, pois ela será executada a cada iteração do loop. Ao definir um breakpoint no código dentro da instrução if, o debugger fará uma pausa somente quando a execução entrar na cláusula if.

A linha com o breakpoint será destacada em amarelo no editor de arquivo. Quando for executado no debugger, o programa iniciará com uma pausa na primeira linha, como sempre. Porém, se você clicar em Go, o programa executará em velocidade máxima até alcançar a linha com o breakpoint definido. Então você poderá clicar em Go, Over, Step ou Out para continuar normalmente.

Se quiser remover um breakpoint, clique com o botão direito do mouse na linha do editor de arquivo e selecione Clear Breakpoint (Remover breakpoint) no menu. O destaque em amarelo desaparecerá e o debugger não fará uma pausa nessa linha no futuro.

Resumo

As asserções, as exceções, o logging e o debugger são ferramentas valiosas para encontrar e evitar bugs em seu programa. As asserções com a instrução assert do Python são uma ótima maneira de implementar "verificações de sanidade" que avisarão com antecedência quando uma condição necessária não for verdadeira. As asserções servem somente para os erros dos quais o programa não deverá tentar se recuperar e devem falhar rapidamente. Caso contrário, utilize uma exceção.

Uma exceção pode ser capturada e tratada pelas instruções try e except. O módulo logging é uma ótima maneira de olhar seu código enquanto ele estiver executando e é muito mais conveniente do que usar a função print() por causa dos seus diferentes níveis de logging e da capacidade de fazer log em um arquivo-texto.

O debugger permite executar seu programa passo a passo, uma linha de cada vez. De modo alternativo, seu programa poderá ser executado em velocidade normal e o debugger poderá fazer uma pausa na execução sempre que alcançar uma linha com um breakpoint definido. Ao usar o debugger, podemos ver o estado de qualquer variável em qualquer ponto do tempo de

vida do programa.

Essas ferramentas e técnicas de debugging ajudarão você a criar programas que funcionem. Introduzir bugs acidentalmente em seu código é um fato da vida, independentemente de quantos anos de experiência de codificação você possa ter.

Exercícios práticos

1. Escreva uma instrução assert que dispare um AssertionError se a variável spam for um inteiro menor do que 10.
2. Escreva uma instrução assert que dispare um AssertionError se as variáveis eggs e bacon contiverem strings que sejam iguais, sem considerar a diferença entre letras maiúsculas e minúsculas (ou seja, 'hello' e 'hello' são consideradas iguais, assim como 'goodbye' e 'GOODbye' também são consideradas iguais).
3. Escreva uma instrução assert que *sempre* dispare um AssertionError.
4. Quais são as duas linhas que seu programa deve ter para poder chamar logging.debug()?
5. Quais são as duas linhas que seu programa deve ter para fazer logging.debug() enviar uma mensagem de logging para um arquivo chamado *programLog.txt*?
6. Quais são os cinco níveis de logging?
7. Qual linha de código pode ser adicionada para desabilitar todas as mensagens de logging em seu programa?
8. Por que o uso de mensagens de logging é melhor que utilizar print() para exibir a mesma mensagem?
9. Quais são as diferenças entre os botões Step, Over e Out da janela Debug Control?
10. Após clicar em Go na janela Debug Control, em que momento o debugger para?
11. O que é um breakpoint?
12. Como um breakpoint é definido em uma linha de código no IDLE?

Projeto prático

Para exercitar, escreva um programa que execute a tarefa a seguir.

Debugging em um programa de lançamento de moeda

O programa a seguir foi criado para ser um jogo simples de adivinhação para o lançamento de uma moeda. O jogador pode dar dois palpites (é um jogo fácil). Entretanto o programa contém diversos bugs. Execute o programa algumas vezes para encontrar os bugs que impedem o programa de funcionar corretamente.

```
import random
guess = ''
while guess not in ('heads', 'tails'):
    print('Guess the coin toss! Enter heads or tails:')
    guess = input()
toss = random.randint(0, 1) # 0 é coroa (tails), 1 é cara (heads)
if toss == guess:
    print('You got it!')
else:
    print('Nope! Guess again!')
    guesss = input()
    if toss == guess:
        print('You got it!')
    else:
        print('Nope. You are really bad at this game.')
```

CAPÍTULO 11

WEB SCRAPING

Naqueles raros e terríveis momentos em que estou sem Wi-Fi, percebo o quanto do que faço em meu computador é realmente o que faço na Internet. Por puro hábito, vejo-me tentando verificar emails, ler os feeds de meus amigos no Twitter ou responder à pergunta “Kurtwood Smith teve algum papel importante antes de atuar no *RoboCop* original em 1987?”.[1]

Como muitos dos trabalhos em um computador envolvem acessar a Internet, seria ótimo se os seus programas pudessem fazer acessos online. *Web scraping* é um termo para se referir ao uso de um programa para fazer download e processar conteúdos da Web. Por exemplo, o Google executa diversos programas de web scraping para indexar páginas web em sua ferramenta de pesquisa. Neste capítulo, conheceremos diversos módulos que facilitam a extração de informações de páginas web em Python.

webbrowser Vem junto com o Python e abre um navegador em uma página específica.

Requests Faz downloads de arquivos e de páginas web da Internet.

Beautiful Soup Faz parse de HTML, que é o formato em que as páginas web são escritas.

Selenium Inicia e controla um navegador web. O Selenium é capaz de preencher formulários e simular cliques de mouse nesse navegador.

Projeto: mapIt.py com o módulo webbrowser

A função open() do módulo webbrowser pode iniciar um novo navegador em um URL especificado. Digite o seguinte no shell interativo:

```
>>> import webbrowser
>>> webbrowser.open('http://inventwithpython.com/')
```

Uma aba do navegador web será aberta com o URL

http://inventwithpython.com/. É praticamente a única tarefa que o módulo webbrowser faz. Mesmo assim, a função open() possibilita realizar algumas tarefas interessantes. Por exemplo, copiar um endereço residencial para o clipboard (área de transferência) e apresentar um mapa do Google Maps com esse endereço é uma tarefa maçante. Você poderia eliminar alguns passos dessa tarefa ao escrever um script simples para iniciar automaticamente o mapa em seu navegador usando o conteúdo de seu clipboard. Dessa maneira, será necessário apenas copiar o endereço para o clipboard e executar o script; o mapa será carregado para você.

Eis o que o seu programa deve fazer:

- Obter um endereço a partir dos argumentos da linha de comando ou do clipboard.
- Abrir o navegador web com a página do Google Maps para o endereço.

Isso significa que seu código deverá fazer o seguinte:

- Ler os argumentos de linha de comando em sys.argv.
- Ler o conteúdo do clipboard.
- Chamar a função webbrowser.open() para abrir o navegador web.

Abra uma nova janela no editor de arquivo e salve esse arquivo como *mapIt.py*.

Passo 1: Identificar o URL

De acordo com as instruções no apêndice B, configure *mapIt.py* para que, ao ser executado a partir da linha de comando desta maneira

```
C:\> mapit 870 Valencia St, San Francisco, CA 94110
```

o script utilize os argumentos da linha de comando no lugar do clipboard. Se não houver nenhum argumento na linha de comando, o programa saberá que deve usar o conteúdo do clipboard.

Inicialmente, será preciso descobrir o URL a ser usado para um dado endereço. Ao carregar *http://maps.google.com/* no navegador e procurar um endereço, o URL na barra de endereço terá o seguinte aspecto: *https://www.google.com/maps/place/870+Valencia+St/@37.7590311,-122.42*

O endereço está no URL, porém há bastante texto adicional presente

também. Com frequência, os sites acrescentam dados aos URLs para ajudar a monitorar os visitantes ou personalizar os sites. No entanto, se você simplesmente tentar acessar *https://www.google.com/maps/place/870+Valencia+St+San+Francisco+CA/*, descobrirá que a página correta será apresentada. Sendo assim, seu programa pode ser definido para abrir um navegador web em 'https://www.google.com/maps/place/*string_com_seu_endereço*' (em que *string_com_seu_endereço* é o endereço que você quer no mapa).

Passo 2: Tratar argumentos da linha de comando

Faça o seu código ter o seguinte aspecto:

```
#! python3
# mapIt.py – Inicia um mapa no navegador usando um endereço da
# linha de comando ou do clipboard.

import webbrowser, sys
if len(sys.argv) > 1:
    # Obtém o endereço da linha de comando.
    address = ' '.join(sys.argv[1:])

# TODO: Obtém o endereço do clipboard.
```

Após a linha shebang #! do programa, importe o módulo webbrowser para iniciar o navegador e o módulo sys para ler os argumentos da linha de comando em potencial. A variável sys.argv armazena uma lista contendo o nome de arquivo do programa e os argumentos da linha de comando. Se essa lista contiver mais informações além do nome do arquivo, len(sys.argv) será avaliado como um inteiro maior do que 1, o que quer dizer que argumentos de linha de comando foram fornecidos.

Os argumentos da linha de comando normalmente são separados por espaços, porém, nesse caso, você deverá interpretar todos os argumentos como uma única string. Como sys.argv é uma lista de strings, podemos passá-la para o método join(), que retornará um único valor de string. Não queremos que o nome do programa esteja nessa string, portanto, em vez de passar sys.argv, passamos sys.argv[1:] para desconsiderar o primeiro elemento do

array. A string final para a qual essa expressão é avaliada é armazenada na variável address.

Se o seu programa for executado com o seguinte na linha de comando:

```
mapit 870 Valencia St, San Francisco, CA 94110
```

a variável sys.argv conterá o valor de lista a seguir:

```
['mapIt.py', '870', 'Valencia', 'St, ', 'San', 'Francisco, ', 'CA', '94110']
```

A variável address conterá a string '870 Valencia St, San Francisco, CA 94110'.

Passo 3: Tratar o conteúdo do clipboard e iniciar o navegador

Faça o seu código ter o seguinte aspecto:

```
#! python3
# mapIt.py – Inicia um mapa no navegador usando um endereço da
# linha de comando ou do clipboard.

import webbrowser, sys, pyperclip
if len(sys.argv) > 1:
    # Obtém o endereço da linha de comando.
    address = ' '.join(sys.argv[1:])
else:
    # Obtém o endereço do clipboard.
    address = pyperclip.paste()

webbrowser.open('https://www.google.com/maps/place/' + address)
```

Se não houver nenhum argumento na linha de comando, o programa suporá que o endereço está armazenado no clipboard. O conteúdo do clipboard pode ser obtido com pyperclip.paste() e será armazenado em uma variável chamada address. Por fim, para iniciar o navegador web com o URL do Google Maps, chame webbrowser.open().

Embora alguns dos programas que você criará realizarão tarefas enormes que farão você economizar horas, usar um programa que o faça economizar convenientemente alguns segundos sempre que uma tarefa comum for realizada, por exemplo, obter um mapa com um endereço, será igualmente

satisfatório. A tabela 11.1 compara os passos necessários para exibir um mapa com e sem *mapIt.py*.

Tabela 11.1 – Obtendo um mapa com e sem mapIt.py

Obtendo um mapa manualmente	Usando *mapIt.py*
Marque o endereço. Copie o endereço. Abra o navegador web. Acesse *http://maps.google.com/*. Clique no campo de texto para o endereço. Cole o endereço. Tecle ENTER.	Marque o endereço. Copie o endereço. Execute *mapIt.py*.

Você percebeu como *mapIt.py* deixa essa tarefa menos tediosa?

Ideias para programas semelhantes

Desde que você tenha um URL, o módulo webbrowser permite que os usuários eliminem o passo referente a abrir o navegador e acessar um site. Outros programas poderão usar essa funcionalidade para fazer o seguinte:

• Abrir todos os links de uma página em abas separadas do navegador.

• Abrir o navegador com o URL para a previsão de tempo local.

• Abrir vários sites de redes sociais que você verifica regularmente.

Fazendo download de arquivos da Web com o módulo requests

O módulo requests permite fazer facilmente o download de arquivos da Web sem se preocupar com problemas complicados como erros de rede, problemas de conexão e compressão de dados. O módulo requests não vem com o Python, portanto será necessário instalá-lo antes. Na linha de comando, execute **pip install requests**. (O Apêndice A contém detalhes adicionais sobre a instalação de módulos de terceiros.)

O módulo requests foi criado porque o módulo urllib2 do Python é complicado demais para usar. Com efeito, pegue um marcador permanente e risque todo este parágrafo. Esqueça que cheguei a mencionar o urllib2. Se precisar fazer download de algo da Web, basta usar o módulo requests.

A seguir, faça um teste simples para garantir que o módulo requests foi instalado corretamente. Digite o seguinte no shell interativo:

```
>>> import requests
```

Se nenhuma mensagem de erro for apresentada, é sinal de que o módulo requests foi instalado com sucesso.

Fazendo download de uma página web com a função requests.get()

A função requests.get() aceita uma string contendo um URL para download. Ao chamar type() no valor de retorno de requests.get(), você perceberá que um objeto Response é retornado contendo a resposta que o servidor web forneceu para a sua solicitação. Explicarei o objeto Response com mais detalhes posteriormente, mas, por enquanto, digite o seguinte no shell interativo enquanto seu computador estiver conectado à Internet:

```
  >>> import requests
u >>> res = requests.get('https://automatetheboringstuff.com/files/rj.txt')
  >>> type(res)
  <class 'requests.models.Response'>
v >>> res.status_code == requests.codes.ok
  True
  >>> len(res.text)
  178981
  >>> print(res.text[:250])
  The Project Gutenberg EBook of Romeo and Juliet, by William Shakespeare

  This eBook is for the use of anyone anywhere at no cost and with
  almost no restrictions whatsoever.  You may copy it, give it away or
  re-use it under the terms of the Proje
```

O URL acessa uma página web textual contendo toda a peça *Romeo and Juliet* (Romeu e Julieta) disponibilizada no site do livro u. Podemos dizer que a solicitação dessa página web foi bem-sucedida verificando o atributo status_code do objeto Response. Se o seu valor for igual a requests.codes.ok, então tudo correu bem v. [A propósito, o código de status para “OK” no protocolo HTTP é 200. Talvez você já tenha familiaridade com o código de status 404

para “Not Found” (Não encontrado).]

Se a solicitação for bem-sucedida, a página web baixada será armazenada como uma string na variável text do objeto Response. Essa variável armazena uma string extensa contendo toda a peça; a chamada a len(res.text) mostra que ela tem mais de 178.000 caracteres. Por fim, chamar print(res.text[:250]) exibe somente os primeiros 250 caracteres.

Verificando se houve erros

Como vimos, o objeto Response tem um atributo status_code que pode ser comparado a requests.codes.ok para testar se o download foi bem-sucedido. Uma maneira mais simples de verificar se houve sucesso consiste em chamar o método raise_for_status() no objeto Response. Isso fará uma exceção ser gerada caso tenha havido um erro no download do arquivo e não fará nada se o download for bem-sucedido. Digite o seguinte no shell interativo:

```
>>> res = requests.get('http://inventwithpython.com/page_that_does_not_exist')
>>> res.raise_for_status()
Traceback (most recent call last):
  File "<pyshell#138>", line 1, in <module>
    res.raise_for_status()
  File "C:\Python34\lib\site-packages\requests\models.py", line 773, in raise_for_status
    raise HTTPError(http_error_msg, response=self)
requests.exceptions.HTTPError: 404 Client Error: Not Found
```

O método raise_for_status() é uma ótima maneira de garantir que um programa seja interrompido caso um download indevido ocorra. Isso é bom: você deve interromper seu programa assim que algum erro inesperado ocorrer. Se um download com falhas *não* for motivo para interromper o seu programa, você poderá inserir instruções try e except ao redor de sua linha raise_for_status() para que esse erro seja tratado sem provocar falhas.

```
import requests
res = requests.get('http://inventwithpython.com/page_that_does_not_exist')
try:
    res.raise_for_status()
except Exception as exc:
```

```
print('There was a problem: %s' % (exc))
```

Essa chamada ao método raise_for_status() fará o programa gerar a saída a seguir:

```
There was a problem: 404 Client Error: Not Found
```

Sempre chame raise_for_status() após chamar requests.get(). Você deve garantir que o download realmente seja bem-sucedido antes que seu programa continue.

Salvando arquivos baixados no disco rígido

A partir de agora, você poderá salvar a página web em um arquivo em seu disco rígido com a função open() e o método write() padrões. Porém há algumas pequenas diferenças. Inicialmente, você deve abrir o arquivo em modo de *escrita binária* passando a string 'wb' como segundo argumento de open(). Mesmo que a página estiver em formato texto simples (como o texto de *Romeu e Julieta* baixado anteriormente), será necessário gravar dados binários em vez de dados em formato texto para preservar a *codificação Unicode* do texto.

CODIFICAÇÕES UNICODE

As codificações Unicode estão além do escopo deste livro, porém você poderá conhecê-las melhor nas páginas web a seguir:

- Joel on Software: The Absolute Minimum Every Software Developer Absolutely, Positively Must Know About Unicode and Character Sets (No Excuses!) [Joel sobre software: o mínimo absoluto que todo desenvolvedor de software definitivamente deve saber sobre Unicode e conjuntos de caracteres (sem desculpas!)]: *http://www.joelonsoftware.com/articles/Unicode.html*
- Pragmatic Unicode (Unicode pragmático): *http://nedbatchelder.com/text/unipain.html*

Para gravar a página web em um arquivo, podemos usar um loop for com o método iter_content() do objeto Response.

```
>>> import requests
>>> res = requests.get('https://automatetheboringstuff.com/files/rj.txt')
```

```
>>> res.raise_for_status()
>>> playFile = open('RomeoAndJuliet.txt', 'wb')
>>> for chunk in res.iter_content(100000):
        playFile.write(chunk)

100000
78981
>>> playFile.close()
```

O método iter_content() retorna “porções” do conteúdo a cada iteração pelo loop. Cada porção tem o tipo de dado *bytes* e é possível especificar quantos bytes cada porção terá. Cem mil bytes geralmente é um bom tamanho, portanto passe 100000 como argumento para iter_content().

O arquivo *RomeoAndJuliet.txt* estará agora no diretório de trabalho atual. Observe que, embora o nome do arquivo no site fosse *pg1112.txt*, o arquivo em seu disco rígido terá um nome diferente. O módulo requests simplesmente trata o downloading do conteúdo de páginas web. Depois que o download da página for feito, esse conteúdo será simplesmente um dado em seu programa. Mesmo que você perca sua conexão com a Internet após o download da página web, todos os dados da página continuarão em seu computador.

O método write() retorna a quantidade de bytes gravada no arquivo. No exemplo anterior, havia 100.000 bytes na primeira porção e a parte restante do arquivo tinha somente 78.981 bytes.

Para revisar, eis o processo completo para fazer download de um arquivo e salvá-lo:

1. Chame requests.get() para fazer download do arquivo.
2. Chame open() com 'wb' para criar um novo arquivo em modo de escrita binária.
3. Crie um loop com o método iter_content() do objeto Response.
4. Chame write() a cada iteração para gravar o conteúdo no arquivo.
5. Chame close() para fechar o arquivo.

Isso é tudo sobre o módulo requests! O uso do loop for e de iter_content() podem parecer complicados se comparados ao fluxo de trabalho com open()/write()/close() que usamos para gravar arquivos-texto, porém servem

para garantir que o módulo requests não consuma muita memória mesmo que arquivos enormes sejam baixados. Você poderá conhecer outros recursos do módulo requests em *http://requests.readthedocs.org/*.

HTML

Antes de começar a selecionar páginas web, você deverá conhecer um pouco de HTML básico. Veremos também como acessar as ferramentas eficazes de desenvolvedor em seu navegador web, que facilitarão bastante a extração de informações da Web.

Recursos para aprender HTML

O *HTML* (*Hypertext Markup Language*, ou Linguagem de marcação de hipertexto) é o formato em que as páginas web são escritas. Neste capítulo, vou supor que você tem um pouco de experiência básica com HTML, porém, se houver necessidade de um tutorial para iniciantes, sugiro um dos sites a seguir:

- *http://htmldog.com/guides/html/beginner/*
- *http://www.codecademy.com/tracks/web/*
- *https://developer.mozilla.org/en-US/learn/html/*

Uma revisão rápida

Caso algum tempo já tenha transcorrido desde a última vez que você viu algum HTML, apresentaremos uma rápida visão geral do básico. Um arquivo HTML é um arquivo em formato texto simples com extensão *.html*. O texto nesses arquivos é cercado por *tags*, que são palavras entre sinais de menor e maior. As tags dizem ao navegador como a página web deve ser formatada. Uma tag de abertura e uma tag de fechamento podem conter um texto e compor um *elemento*. O *texto* (ou *HTML interno*) é o conteúdo entre as tags de abertura e de fechamento. Por exemplo, o HTML a seguir exibirá *Hello world!* no navegador com *Hello* em negrito:

```
<strong>Hello</strong> world!
```

Esse HTML terá o aspecto mostrado na figura 11.1 em um navegador.

Figura 11.1 – Hello world! exibido no navegador.

A tag de abertura informa que o texto nela incluído aparecerá em negrito. A tag de fechamento informa o navegador em que ponto o texto em negrito termina.

Há várias tags diferentes em HTML. Algumas dessas tags têm propriedades extras na forma de *atributos* entre os sinais de menor e de maior. Por exemplo, a tag <a> inclui um texto que deverá ser um link. O URL ao qual o texto está associado é determinado pelo atributo href. Aqui está um exemplo:

```
Al's free <a href="http://inventwithpython.com">Python books</a>.
```

Esse HTML terá o aspecto mostrado na figura 11.2 em um navegador.

Figura 11.2 – O link exibido no navegador.

Alguns elementos têm um atributo id usado para identificar unicamente o elemento na página. Com frequência, você instruirá seus programas a procurar um elemento pelo seu atributo id, portanto descobrir o atributo id de um elemento usando as ferramentas de desenvolvedor do navegador será uma tarefa comum ao escrever programas de web scraping.

Visualizando o código-fonte HTML de uma página web

Será necessário observar o código-fonte HTML das páginas web com as quais seus programas trabalharão. Para isso, clique com o botão direito do mouse (ou dê um CTRL-clique no OS X) em qualquer página web em seu navegador e selecione View Source (Exibir código-fonte) ou View page

source (Exibir código-fonte da página) para ver o texto HTML da página (observe a figura 11.3). Esse é o texto que seu navegador realmente recebe. O navegador sabe como exibir, ou seja, *renderizar*, a página web a partir desse HTML.

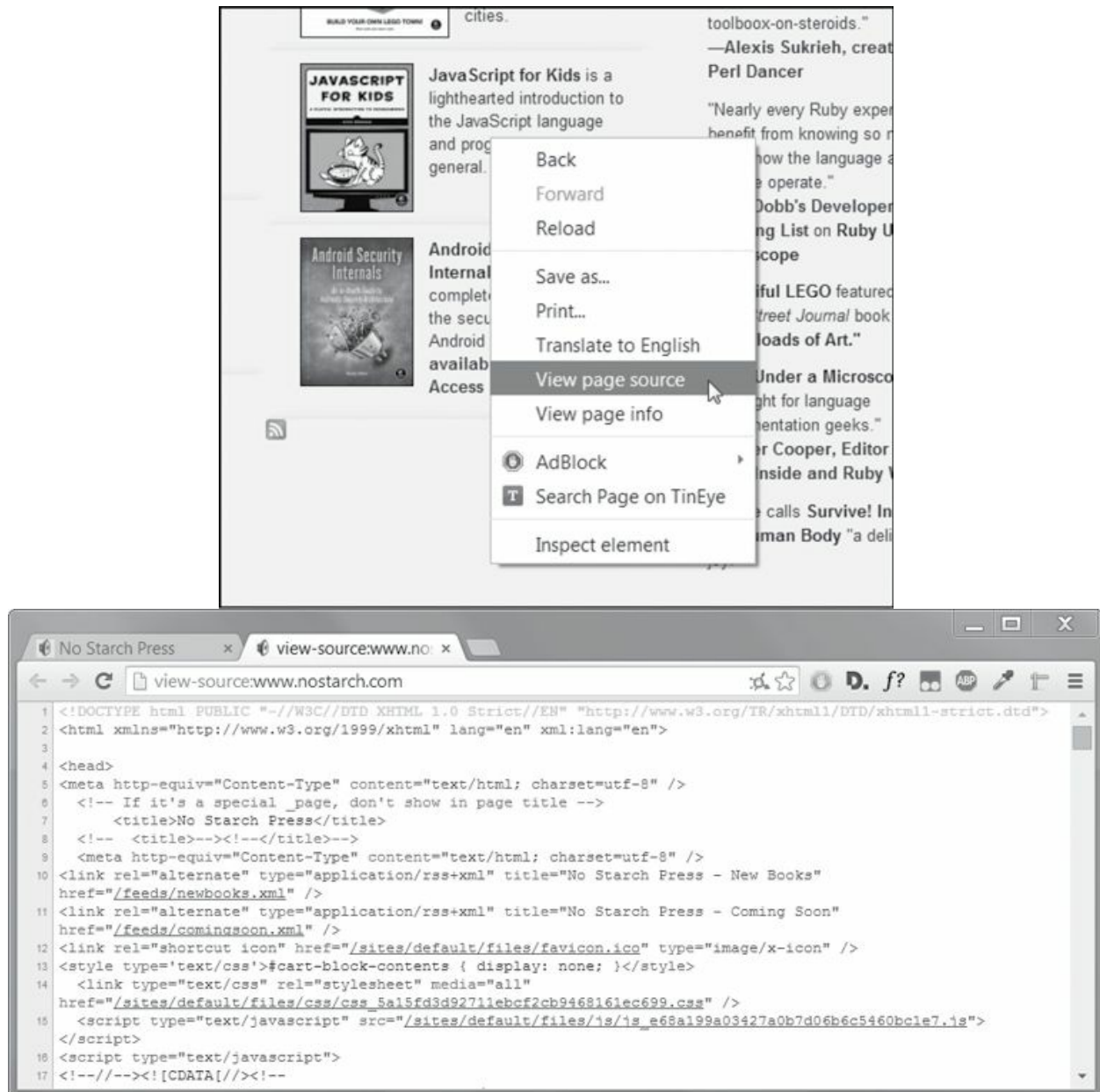
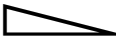



Figura 11.3 – Visualizando o código-fonte de uma página web.

Recomendo altamente visualizar o código HTML de alguns de seus sites favoritos. Não há problemas se você não entender totalmente o que estiver vendo ao observar o código-fonte. Não será preciso dominar completamente

o HTML para escrever programas simples de web scraping – afinal de contas, você não estará criando seus próprios sites. Será necessário apenas ter conhecimento suficiente para selecionar os dados de um site existente.

Abrindo as ferramentas de desenvolvedor em seu navegador

Além de visualizar o código-fonte de uma página web, você poderá ver o HTML de uma página usando as ferramentas de desenvolvedor em seu navegador. No Chrome e no Internet Explorer para Windows, as ferramentas de desenvolvedor já estão instaladas e você poderá pressionar F12 para que elas sejam apresentadas (veja a figura 11.4). Se pressionar F12 novamente, as ferramentas de desenvolvedor desaparecerão. No Chrome, você também poderá fazer as ferramentas de desenvolvedor serem exibidas selecionando View4Developer4Developer Tools (Visualizar4Desenvolvedor4Ferramentas do desenvolvedor). No OS X, pressionar -OPTION-I abrirá as Ferramentas do desenvolvedor no Chrome.

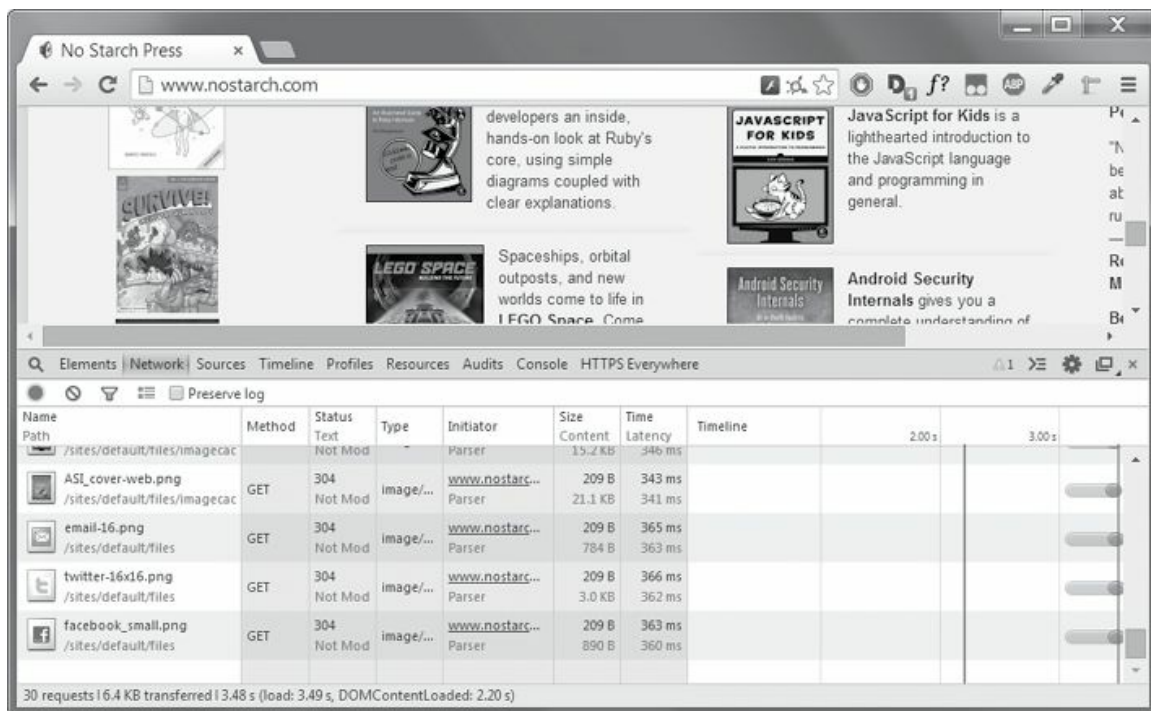
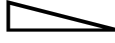
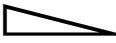



Figura 11.4 – A janela de Ferramentas do Desenvolvedor no navegador Chrome.

No Firefox, o Web Developer Tools Inspector poderá ser apresentado ao pressionar CTRL-SHIFT-C no Windows e no Linux ou ⌘-OPTION-C no OS X. Seu layout é quase idêntico ao das ferramentas de desenvolvedor do Chrome.

No Safari, abra a janela Preferences (Preferências); no painel Advanced (Avançado), marque a opção Show Develop menu in the menu bar (Mostrar menu Desenvolvedor na barra de menus). Após essa opção ter sido habilitada, as ferramentas de desenvolvedor poderão ser apresentadas ao pressionar ⌘-OPTION-I.

Depois de habilitar ou de instalar as ferramentas de desenvolvedor em seu navegador, você poderá clicar com o botão direito do mouse em qualquer parte da página web e selecionar Inspect Element (Inspecionar elemento) no menu de contexto para que o HTML responsável por essa parte da página seja apresentado. Isso será útil quando você começar a fazer parse do HTML em seus programas de web scraping.

NÃO UTILIZE EXPRESSÕES REGULARES PARA PARSE DE HTML

Localizar uma porção específica de HTML em uma string parece ser a ocasião perfeita para usar expressões regulares. Entretanto não aconselho a fazer isso. Há muitas maneiras diferentes de o HTML ser formatado e continuar sendo considerado como válido, porém tentar capturar todas essas possíveis variações em uma expressão regular poderá ser tedioso e estará suscetível a erros. Um módulo desenvolvido especificamente para parsing de HTML, por exemplo, o Beautiful Soup, terá menos chances de resultar em bugs.

Você poderá encontrar uma argumentação mais detalhada para o motivo pelo qual você não deve fazer parse de HTML usando expressões regulares em *http://stackoverflow.com/a/1732454/1893164/*.

Usando as ferramentas de desenvolvedor para encontrar elementos HTML

Depois que seu programa fizer o download de uma página web usando o módulo requests, você terá o conteúdo HTML da página na forma de um valor único de string. Agora devemos descobrir qual parte do HTML corresponde à

informação na página web em que você está interessado.

É nesse caso que as ferramentas de desenvolvedor do navegador poderão ajudar. Suponha que você queira criar um programa para extrair dados de previsão do tempo de *http://weather.gov/*. Antes de escrever qualquer código, faça uma pequena pesquisa. Se você acessar o site e procurar o CEP 94105, o site conduzirá você a uma página que mostrará a previsão do tempo para essa região.

O que deverá acontecer se você estiver interessado em extrair informações de temperatura para esse CEP? Clique com o botão direito do mouse no local em que está essa informação na página (ou dê um CONTROL-clique no OS X) e selecione Inspect Element (Inspecionar elemento) no menu de contexto apresentado. Isso fará a janela de Ferramentas do Desenvolvedor ser apresentada, exibindo o HTML que gera essa parte da página web em particular. A figura 11.5 mostra as ferramentas de desenvolvedor abertas com o HTML para a temperatura.

A partir das ferramentas de desenvolvedor, você poderá ver que o HTML responsável para a parte contendo a temperatura na página web é `<p class="myforecast-current-lrg">59°F</p>`. É exatamente isso que estávamos procurando! Parece que a informação sobre temperatura está contida em um elemento `<p>` cuja classe é `myforecast-current-lrg`. Agora que identificamos o que estávamos procurando, o módulo `BeautifulSoup` ajudará a encontrar essa informação na string.

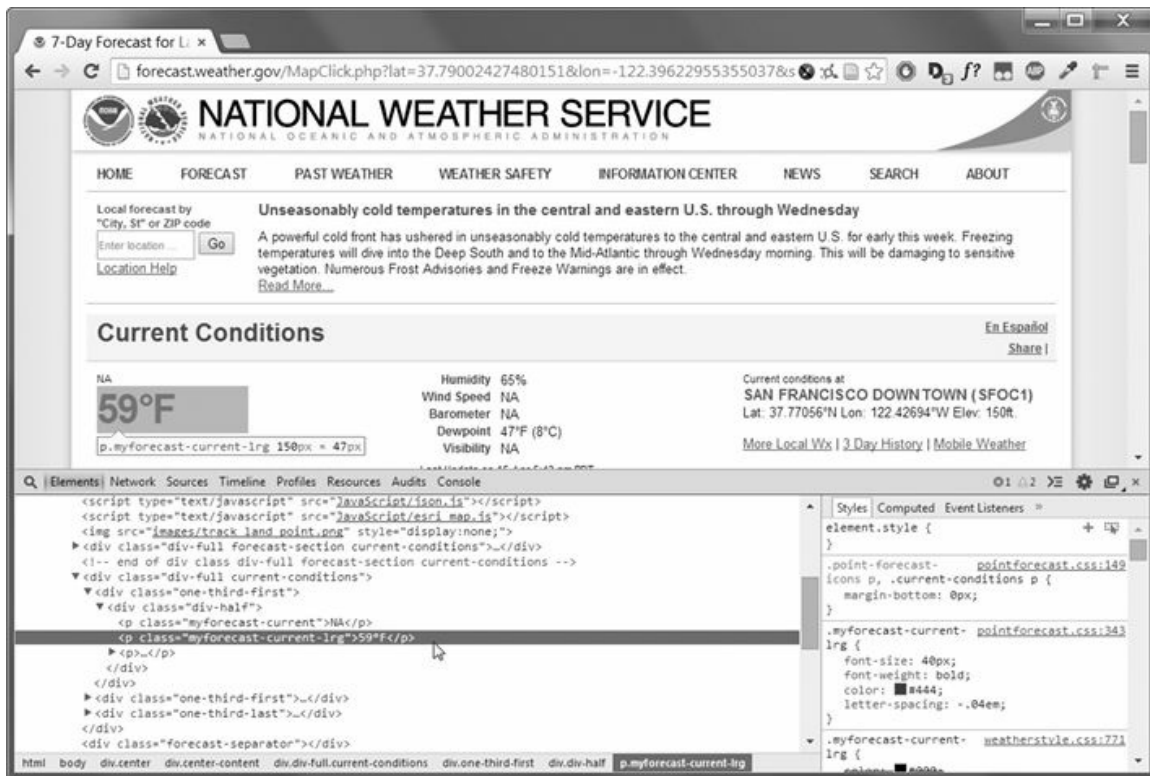



Figura 11.5 – Inspecionando o elemento que contém o texto referente à temperatura usando as ferramentas de desenvolvedor.

Fazendo parse de HTML com o módulo BeautifulSoup

O Beautiful Soup é um módulo usado para extrair informações de uma página HTML (e é muito melhor para isso do que as expressões regulares). O nome do módulo BeautifulSoup é bs4 (de Beautiful Soup versão 4). Para instalá-lo, será necessário executar pip install beautifulsoup4 na linha de comando. (Dê uma olhada no Apêndice A para obter instruções sobre a instalação de módulos de terceiros.) Embora beautifulsoup4 seja o nome usado na instalação, para importar o Beautiful Soup, execute import bs4.

Neste capítulo, os exemplos com Beautiful Soup farão *parse* (ou seja, analisarão e identificarão partes) de um arquivo HTML no disco rígido. Abra uma nova janela no editor de arquivo do IDLE, digite o programa a seguir e salve-o como *example.py*. De modo alternativo, você poderá fazer seu download a partir de *http://nostarch.com/automatestuff/*.

```
<!-- Este é o arquivo example.html de exemplo. -->
```

```
<html><head><title>The Website Title</title></head>
<body>
<p>Download my <strong>Python</strong> book from <a
href="http://inventwithpython.com">my website</a>.</p>
<p class="slogan">Learn Python the easy way!</p>
<p>By <span id="author">Al Sweigart</span></p>
</body></html>
```

Como podemos ver, mesmo um arquivo HTML simples envolve muitas tags e atributos diferentes, e a situação começa a ficar rapidamente confusa em sites complexos. Felizmente, o Beautiful Soup facilita bastante trabalhar com HTML.

Criando um objeto BeautifulSoup a partir do HTML

A função bs4.BeautifulSoup() deve ser chamada com uma string contendo o HTML em que o parse será feito. Essa função retorna um objeto BeautifulSoup. Digite o seguinte no shell interativo enquanto seu computador estiver conectado à Internet:

```
>>> import requests, bs4
>>> res = requests.get('http://nostarch.com')
>>> res.raise_for_status()
>>> noStarchSoup = bs4.BeautifulSoup(res.text)
>>> type(noStarchSoup)
<class 'bs4.BeautifulSoup'>
```

Esse código utiliza requests.get() para fazer download da página principal do site da No Starch Press e, em seguida, passa o atributo text da resposta para bs4.BeautifulSoup(). O objeto BeautifulSoup retornado é armazenado em uma variável chamada noStarchSoup.

Você também pode carregar um arquivo HTML de seu disco rígido passando um objeto File para bs4.BeautifulSoup(). Digite o seguinte no shell interativo (certifique-se de que o arquivo *example.html* está no diretório de trabalho):

```
>>> exampleFile = open('example.html')
>>> exampleSoup = bs4.BeautifulSoup(exampleFile)
>>> type(exampleSoup)
<class 'bs4.BeautifulSoup'>
```

Depois que tiver um objeto BeautifulSoup, você poderá usar seus métodos para localizar partes específicas de um documento HTML.

Encontrando um elemento com o método select()

Podemos obter um elemento de uma página web a partir de um objeto BeautifulSoup chamando o método select() e passando uma string com um *seletor* CSS para o elemento que estamos procurando. Os seletores são como expressões regulares: eles especificam um padrão a ser procurado – nesse caso, em páginas HTML em vez de strings de texto genéricas.

Uma discussão completa sobre a sintaxe de seletores CSS está além do escopo deste livro (há um bom tutorial sobre seletores nos recursos em *http://nostarch.com/automatestuff/*), porém aqui está uma breve introdução aos seletores. A tabela 11.2 mostra exemplos dos padrões mais comuns de seletores CSS.

Tabela 11.2 – Exemplos de seletores CSS

Seletor passado ao método select()	Corresponde a...
soup.select('div')	Todos os elementos de nome <div>.
soup.select('#author')	O elemento com um atributo id igual a author.
soup.select('.notice')	Todos os elementos que utilizem um atributo class de CSS chamado notice.
soup.select('div span')	Todos os elementos de nome <span> que estejam em um elemento chamado <div>.
soup.select('div > span')	Todos os elementos de nome <span> que estejam *diretamente* em um elemento chamado <div>, sem que haja outros elementos intermediários.
soup.select('input[name]')	Todos os elementos de nome <input> que tenham um atributo name com qualquer valor.
soup.select('input[type="button"]')	Todos os elementos de nome <input> que tenham um atributo chamado type com o valor button.

Os diversos padrões de seletores podem ser combinados para fazer correspondências mais sofisticadas. Por exemplo, soup.select('p #author') corresponderá a qualquer elemento que tenha um atributo id igual a author, desde que também esteja dentro de um elemento <p>.

O método select() retornará uma lista de objetos Tag, que é o modo como o Beautiful Soup representa um elemento HTML. A lista conterá um objeto Tag para cada correspondência feita no HTML do objeto BeautifulSoup. Os valores de tag podem ser passados para a função str() para que as tags HTML que representam possam ser mostradas. Os valores de tag também têm um atributo attrs que mostra todos os atributos HTML da tag na forma de um dicionário. Usando o arquivo *example.html* anterior, digite o seguinte no shell interativo:

```
>>> import bs4
>>> exampleFile = open('example.html')
>>> exampleSoup = bs4.BeautifulSoup(exampleFile.read())
>>> elems = exampleSoup.select('#author')
>>> type(elems)
<class 'list'>
>>> len(elems)
1
>>> type(elems[0])
<class 'bs4.element.Tag'>
>>> elems[0].getText()
'Al Sweigart'
>>> str(elems[0])
'<span id="author">Al Sweigart</span>'
>>> elems[0].attrs
{'id': 'author'}
```

Esse código extrairá o elemento com id="author" de nosso HTML de exemplo. Usamos select('#author') para retornar uma lista contendo todos os elementos com id="author". Armazenamos essa lista de objetos Tag na variável elems e len(elems) nos informa que há um objeto Tag na lista, ou seja, houve uma correspondência. Chamar getText() no elemento fará o texto desse elemento ser retornado, isto é, o HTML interno. O texto de um elemento é o conteúdo entre as tags de abertura e de fechamento: nesse caso, é 'Al Sweigart'.

Passar o elemento para str() fará uma string ser retornada com as tags de abertura e de fechamento e o texto do elemento. Por fim, attrs nos fornece um dicionário com o atributo 'id' do elemento e o valor desse atributo, ou seja, 'author'.

Podemos também extrair todos os elementos <p> do objeto BeautifulSoup. Digite o seguinte no shell interativo:

```
>>> pElems = exampleSoup.select('p')
>>> str(pElems[0])
'<p>Download my <strong>Python</strong> book from <a
href="http://inventwithpython.com">my website</a>.</p>'
>>> pElems[0].getText()
'Download my Python book from my website.'
>>> str(pElems[1])
'<p class="slogan">Learn Python the easy way!</p>'
>>> pElems[1].getText()
'Learn Python the easy way!'
>>> str(pElems[2])
'<p>By <span id="author">Al Sweigart</span></p>'
>>> pElems[2].getText()
'By Al Sweigart'
```

Desta vez, select() nos forneceu uma lista com três correspondências, que armazenamos em pElems. Usar str() em pElems[0], pElems[1] e pElems[2] mostra cada elemento na forma de uma string, enquanto usar getText() em cada elemento exibe o seu texto.

Obtendo dados dos atributos de um elemento

O método get() de objetos Tag facilita acessar valores de atributos de um elemento. O método recebe uma string contendo um nome de atributo e retorna o valor desse atributo. Usando *example.html*, digite o seguinte no shell interativo:

```
>>> import bs4
>>> soup = bs4.BeautifulSoup(open('example.html'))
>>> spanElem = soup.select('span')[0]
>>> str(spanElem)
'<span id="author">Al Sweigart</span>'
>>> spanElem.get('id')
'author'
>>> spanElem.get('some_nonexistent_addr') == None
True
>>> spanElem.attrs
```

{'id': 'author'}

Nesse caso, usamos select() para encontrar todos os elementos <span> e então armazenamos o primeiro elemento correspondente em spanElem. Passar o nome de atributo 'id' para get() faz o valor do atributo, ou seja, 'author', ser retornado.

Projeto: Pesquisa "Estou com sorte" no Google

Sempre que pesquiso um assunto no Google, não olho apenas um resultado da pesquisa de cada vez. Ao clicar em um link de resultado da pesquisa com o botão do meio do mouse (ou clicar enquanto CTRL estiver pressionado), abro os primeiros links em uma porção de novas abas para ler depois. Faço pesquisas no Google com tanta frequência que esse fluxo de trabalho – abrir meu navegador, pesquisar um assunto e clicar em diversos links, um por um, com o botão do meio do mouse – é tedioso. Seria interessante se eu pudesse simplesmente digitar um termo de pesquisa na linha de comando e fazer meu computador abrir automaticamente um navegador com todos os principais resultados da pesquisa em novas abas. Vamos criar um script para fazer isso.

Eis o que o seu programa deve fazer:

- Obter palavras-chave para pesquisa a partir dos argumentos da linha de comando.
- Obter a página com os resultados da pesquisa.
- Abrir uma aba do navegador para cada resultado.

Isso significa que seu código deverá fazer o seguinte:

- Ler os argumentos da linha de comando em sys.argv.
- Acessar a página com os resultados da pesquisa usando o módulo requests.
- Encontrar os links para cada resultado da pesquisa.
- Chamar a função webbrowser.open() para abrir o navegador web.

Abra uma nova janela no editor de arquivo e salve esse arquivo como *lucky.py*.

Passo 1: Obter os argumentos da linha de comando e solicitar a página de pesquisa

Antes de codificar algo, inicialmente é preciso conhecer o URL da página com os resultados da pesquisa. Ao observar a barra de endereço do navegador após fazer uma pesquisa no Google, podemos ver que a página de resultados tem um URL semelhante a *https://www.google.com/search?q=TERMO_DA_PESQUISA_AQUI*. O módulo requests pode fazer download dessa página e, em seguida, o Beautiful Soup poderá ser usado para encontrar os links dos resultados da pesquisa no HTML. Por fim, você usará o módulo webbrowser para abrir esses links nas abas do navegador.

Faça o seu código ter o seguinte aspecto:

```
#! python3
# lucky.py – Abre vários resultados de pesquisa no Google.

import requests, sys, webbrowser, bs4

print('Googling...')    # exibe um texto enquanto faz download da página do Google
res = requests.get('http://google.com/search?q=' + ' '.join(sys.argv[1:]))
res.raise_for_status()

# TODO: Obtém os links dos principais resultados da pesquisa.

# TODO: Abre uma aba do navegador para cada resultado.
```

O usuário especificará os termos da pesquisa usando argumentos da linha de comando quando o programa for iniciado. Esses argumentos serão armazenados como strings em uma lista em sys.argv.

Passo 2: Encontrar todos os resultados

Agora você deverá usar o Beautiful Soup para extrair os links dos principais resultados da pesquisa do HTML baixado. Mas como descobrimos qual é o seletor correto para essa tarefa? Por exemplo, não podemos simplesmente procurar todas as tags <a>, pois há muitos links em que não estaremos interessados no HTML. Em vez disso, devemos inspecionar a página com os resultados da pesquisa com as ferramentas de desenvolvedor do navegador e tentar encontrar um seletor que escolha somente os links desejados.

Após fazer uma pesquisa no Google em busca de *Beautiful Soup*, você

poderá abrir as ferramentas de desenvolvedor do Google e inspecionar alguns dos elementos que contêm links na página. Eles parecerão extremamente complicados e serão semelhantes a: Beautiful Soup: We called him Tortoise because he taught us..

Não importa se o elemento pareça ser extremamente complicado. Você só precisará encontrar o padrão presente em todos os links do resultado da pesquisa. Porém esse elemento <a> não contém nada que o diferencie facilmente dos elementos <a> que não fazem parte do resultado da pesquisa na página.

Faça o seu código ter o seguinte aspecto:

```
#! python3
# lucky.py – Abre vários resultados de pesquisa no Google.

import requests, sys, webbrowser, bs4

--trecho removido--

# Obtém os links dos principais resultados da pesquisa.
soup = bs4.BeautifulSoup(res.text)

# Abre uma aba do navegador para cada resultado.
linkElems = soup.select('.r a')
```

Porém, se você observar os dados um pouco antes do elemento <a>, verá que há um elemento como este: <h3 class="r">. Ao observar o restante do código-fonte HTML, parece que a classe r é usada somente para links contendo resultados da pesquisa. Não é preciso saber o que é a classe CSS r nem o que ela faz. Só iremos usá-la como um marcador para o elemento <a> que estamos procurando. Podemos criar um objeto BeautifulSoup a partir do texto HTML da página baixada e então usar o seletor '.r a' para encontrar todos os elementos <a> que estejam em um elemento que tenha a classe CSS r.

Passo 3: Abrir abas do navegador web para cada resultado

Por fim, diremos ao programa para que abra abas no navegador web para os nossos resultados. Acrescente o seguinte no final de seu programa:

```
#! python3
# lucky.py – Abre vários resultados de pesquisa no Google.

import requests, sys, webbrowser, bs4

--trecho removido--

# Abre uma aba do navegador para cada resultado.
linkElems = soup.select('.r a')
numOpen = min(5, len(linkElems))
for i in range(numOpen):
    webbrowser.open('http://google.com' + linkElems[i].get('href'))
```

Por padrão, abra os cinco primeiros resultados da pesquisa em novas abas usando o módulo webbrowser. Entretanto o usuário poderá ter feito uma pesquisa que obteve menos de cinco resultados. A chamada a soup.select() retorna uma lista de todos os elementos correspondentes ao seu seletor '.r a', portanto o número de abas que você deseja abrir será 5 ou o tamanho dessa lista (o que for menor).

A função interna min() do Python retorna o menor argumento inteiro ou de ponto flutuante que receber. (Há também uma função max() interna que retorna o maior argumento recebido.) Podemos usar min() para descobrir se há menos de cinco links na lista e armazenar o número de links a serem abertos em uma variável chamada numOpen. Então um loop for que chame range(numOpen) poderá ser executado.

A cada iteração do loop, webbrowser.open() será usado para abrir uma nova aba no navegador web. Observe que o valor do atributo href nos elementos <a> retornados não tem a parte http://google.com inicial, portanto será necessário concatenar esse dado ao valor de string do atributo href.

Agora você poderá abrir instantaneamente os cinco primeiros resultados do Google, por exemplo, para *Python programming tutorials*, ao executar lucky

python programming tutorials na linha de comando! (Veja o Apêndice B para saber como executar programas facilmente em seu sistema operacional.)

Ideias para programas semelhantes

A vantagem de um navegador com abas está no fato de você poder abrir os links facilmente em novas abas para dar uma olhada posteriormente. Um programa que abra diversos links automaticamente de uma só vez pode ser um bom atalho para fazer o seguinte:

- Abrir todas as páginas de produto após pesquisar um site de compras como a Amazon.
- Abrir todos os links para avaliações sobre um único produto.
- Abrir os links de resultados para fotos após realizar uma pesquisa em um site de fotos como o Flickr ou o Imgur.

Projeto: Downloading de todas as tirinhas XKCD

Os blogs e outros sites atualizados regularmente em geral têm uma página inicial com a postagem mais recente, além de um botão Previous (Anterior) na página que conduzirá você à postagem anterior. Então essa postagem também terá um botão Previous e assim sucessivamente, criando um percurso que conduz da página mais recente até a primeira postagem do site. Se quiser uma cópia do conteúdo do site para ler quando não estiver online, você poderá navegar manualmente por todas as páginas e salvar cada uma delas. Contudo esse é um trabalho bem maçante, portanto vamos criar um programa que faça isso.

O XKCD é um webcomic popular para geeks com um site que se enquadra nessa estrutura (veja a figura 11.6). A página inicial em *http://xkcd.com/* contém um botão Prev (Anterior) que conduz o usuário às tirinhas anteriores. Fazer download de cada tirinha manualmente demoraria muito tempo, porém podemos escrever um script para fazer isso em dois minutos.

Eis o que o seu programa deve fazer:

- Carregar a página inicial de XKCD.
- Salvar a imagem da tirinha que está nessa página.

• Seguir o link para Previous Comic (Tirinha anterior).
• Repetir até alcançar a primeira tirinha.

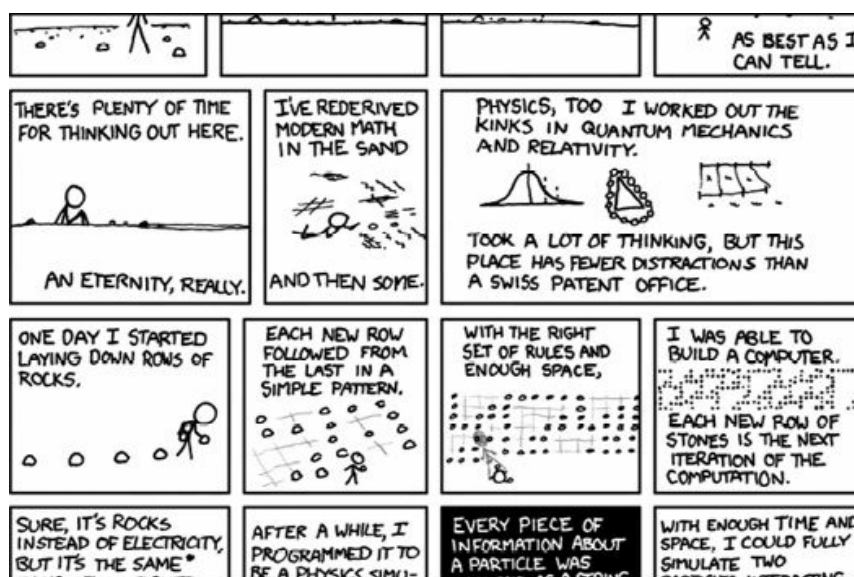



Figura 11.6 – XKCD, "a webcomic of romance, sarcasm, math, and language" (um webcomic com romance, sarcasmo, matemática e linguagem).

Isso significa que seu código deverá fazer o seguinte:

• Fazer download de páginas com o módulo requests.
• Encontrar o URL da imagem da tirinha em uma página usando o Beautiful Soup.
• Fazer download e salvar a imagem da tirinha no disco rígido usando iter_content().
• Encontrar o URL do link Previous Comic (Tirinha anterior) e repetir.

Abra uma nova janela no editor de arquivo e salve esse arquivo como *downloadXkcd.py*.

Passo 1: Design do programa

Se você abrir as ferramentas de desenvolvedor do navegador e inspecionar os elementos da página, encontrará o seguinte:

• O URL do arquivo de imagem da tirinha é dado pelo atributo href de um elemento <img>.
• O elemento <img> está em um elemento <div id="comic">.

- O botão Prev (Anterior) tem um atributo HTML rel cujo valor é prev.
- O botão Prev da primeira tirinha está associado ao URL *http://xkcd.com/#* indicando que não há mais páginas anteriores.

Faça o seu código ter o seguinte aspecto:

```
#! python3
# downloadXkcd.py – Faz download de todas as tirinhas de XKCD.

import requests, os, bs4

url = 'http://xkcd.com'              # url inicial
os.makedirs('xkcd', exist_ok=True)    # armazena as tirinhas em ./xkcd
while not url.endswith('#'):
   # TODO: Faz download da página.

   # TODO: Encontra o URL da imagem da tirinha.

   # TODO: Faz  o download da imagem.

   # TODO: Salva a imagem em ./xkcd.

   # TODO: Obtém o url do botão Prev.
print('Done.')
```

Você terá uma variável url que começa com o valor 'http://xkcd.com'; ela será repetidamente atualizada (em um loop for) com o URL do link Prev da página atual. A cada passo no loop, a tirinha em url será baixada. Você saberá que o loop deve ser encerrado quando url terminar com '#'.

Os arquivos de imagens deverão ser baixados para uma pasta chamada *xkcd* no diretório de trabalho atual. A chamada a os.makedirs() garante que essa pasta exista, e o argumento nomeado exist_ok=True impede que a função lance uma exceção caso essa pasta já exista. O restante do código contém apenas comentários que descrevem o restante de seu programa.

Passo 2: Download da página web

Vamos implementar o código para fazer download da página. Faça o seu código ter o seguinte aspecto:

```
#! python3
# downloadXkcd.py – Faz download de todas as tirinhas de XKCD.

import requests, os, bs4

url = 'http://xkcd.com'              # url inicial
os.makedirs('xkcd', exist_ok=True)    # armazena as tirinhas em ./xkcd
while not url.endswith('#'):
    # Faz download da página.
    print('Downloading page %s...' % url)
    res = requests.get(url)
    res.raise_for_status()

    soup = bs4.BeautifulSoup(res.text)

    # TODO: Encontra o URL da imagem da tirinha.

    # TODO: Faz o download da imagem.

    # TODO: Salva a imagem em ./xkcd.

    # TODO: Obtém o url do botão Prev.
print('Done.')
```

Inicialmente, exiba o url para que o usuário saiba qual URL o programa está prestes a baixar; então utilize a função request.get() do módulo requests para fazer o seu download. Como sempre, você chamará o método raise_for_status() do objeto Response imediatamente para lançar uma exceção e encerrar o programa caso um erro tenha ocorrido no download. Do contrário, um objeto BeautifulSoup será criado a partir do texto da página baixada.

Passo 3: Encontrar e fazer download da imagem da tirinha

Faça o seu código ter o seguinte aspecto:

```
#! python3
# downloadXkcd.py – Faz download de todas as tirinhas de XKCD.

import requests, os, bs4
```

```
--trecho removido--

    # Encontra o URL da imagem da tirinha.
    comicElem = soup.select('#comic img')
    if comicElem == []:
        print('Could not find comic image.')
    else:
        comicUrl = 'http:' + comicElem[0].get('src')
        # Faz o download da imagem.
        print('Downloading image %s...' % (comicUrl))
        res = requests.get(comicUrl)
        res.raise_for_status()

    # TODO: Salva a imagem em ./xkcd.

    # TODO: Obtém o url do botão Prev.

print('Done.')
```

A partir da inspeção da página inicial de XKCD com suas ferramentas de desenvolvedor, sabemos que o elemento <img> para a imagem da tirinha está em um elemento <div> com o atributo id definido com comic, portanto o seletor '#comic img' fará você obter o elemento <img> correto do objeto BeautifulSoup.

Algumas páginas do XKCD têm um conteúdo especial que não é simplesmente um arquivo de imagem. Não há problemas nisso; você poderá simplesmente ignorá-las. Se o seu seletor não encontrar nenhum elemento, soup.select('#comic img') retornará uma lista vazia. Quando isso ocorrer, o programa poderá simplesmente exibir uma mensagem de erro e prosseguir sem fazer o download da imagem.

Caso contrário, o seletor retornará uma lista contendo um elemento <img>. O atributo src poderá ser obtido desse elemento <img> e passado para requests.get() para que o download do arquivo com a imagem da tirinha seja feito.

Passo 4: Salvar a imagem e encontrar a tirinha anterior

Faça o seu código ter o seguinte aspecto:

```
#! python3
# downloadXkcd.py – Faz download de todas as tirinhas de XKCD.

import requests, os, bs4

--trecho removido--

        # Salva a imagem em ./xkcd.
        imageFile = open(os.path.join('xkcd', os.path.basename(comicUrl)), 'wb')
        for chunk in res.iter_content(100000):
            imageFile.write(chunk)
        imageFile.close()

    # Obtém o url do botão Prev.
    prevLink = soup.select('a[rel="prev"]')[0]
    url = 'http://xkcd.com' + prevLink.get('href')

print('Done.')
```

A essa altura, o arquivo de imagem da tirinha está armazenado na variável res. É preciso gravar os dados dessa imagem em um arquivo no disco rígido.

Será necessário ter um nome para o arquivo que conterá a imagem local; esse nome deverá ser passado para open(). comicUrl terá um valor como 'http://imgs.xkcd.com/comics/heartbleed_explanation.png' – que, conforme você deve ter percebido, parece muito com um path de arquivo. Com efeito, podemos chamar os.path.basename() com comicUrl e ele retornará somente a última parte do URL, ou seja, 'heartbleed_explanation.png'. Podemos usar isso como nome do arquivo quando salvarmos a imagem no disco rígido. Junte esse nome ao nome de sua pasta xkcd usando os.path.join() para que seu programa utilize barras invertidas (\) no Windows e barras para frente (/) no OS X e no Linux. Agora que finalmente temos um nome para o arquivo, podemos chamar open() para abrir um novo arquivo em modo 'wb' de "escrita binária".

Lembre-se de que, conforme vimos anteriormente neste capítulo, para salvar arquivos baixados com Requests, devemos percorrer o valor de retorno do método iter_content() com um loop. O código do loop for grava porções de dados da imagem (no máximo 100.000 bytes de cada vez) no arquivo e, em

seguida, o arquivo será fechado. A imagem agora está salva em seu disco rígido.

Depois disso, o seletor 'a[rel="prev"]' identifica o elemento <a> cujo atributo rel está definido com prev e o atributo href desse elemento <a> poderá ser usado para obter o URL da tirinha anterior, que será armazenado em url. Então o loop while começará todo o processo de download novamente para essa tirinha.

A saída desse programa terá a seguinte aparência:

```
Downloading page http://xkcd.com...
Downloading image http://imgs.xkcd.com/comics/phone_alarm.png...
Downloading page http://xkcd.com/1358/...
Downloading image http://imgs.xkcd.com/comics/nro.png...
Downloading page http://xkcd.com/1357/...
Downloading image http://imgs.xkcd.com/comics/free_speech.png...
Downloading page http://xkcd.com/1356/...
Downloading image http://imgs.xkcd.com/comics/orbital_mechanics.png...
Downloading page http://xkcd.com/1355/...
Downloading image http://imgs.xkcd.com/comics/airplane_message.png...
Downloading page http://xkcd.com/1354/...
Downloading image http://imgs.xkcd.com/comics/heartbleed_explanation.png...
--trecho removido--
```

Esse projeto é um bom exemplo de um programa que pode seguir links automaticamente para extrair grandes quantidades de dados da Web. Você pode conhecer os demais recursos do Beautiful Soup consultando sua documentação em *http://www.crummy.com/software/BeautifulSoup/bs4/doc/*.

Ideias para programas semelhantes

Fazer download de páginas e seguir links formam a base de muitos programas de web crawling. Programas semelhantes também poderão:

- Fazer backup de um site completo seguindo todos os seus links.
- Copiar todas as mensagens de um fórum web.
- Duplicar o catálogo de itens à venda em uma loja online.

Os módulos requests e BeautifulSoup são ótimos, desde que você possa determinar o URL que deverá ser passado para requests.get(). Às vezes, porém,

não é tão fácil identificá-lo. Pode ser que o site para o qual você quer que seu programa navegue exija um login antes. O módulo selenium proporcionará aos seus programas a capacidade de realizar essas tarefas sofisticadas.

Controlando o navegador com o módulo Selenium

O módulo selenium permite que o Python controle diretamente o navegador ao clicar em links e preencher informações de login por meio de programação, quase como se houvesse um ser humano interagindo com a página. O Selenium permite interagir com páginas web de uma maneira muito mais sofisticada que o Requests e o Beautiful Soup; contudo, pelo fato de iniciar um navegador web, ele é um pouco mais lento e difícil de ser executado em background se, por exemplo, for necessário fazer download de apenas alguns arquivos da Web.

O Apêndice A contém passos mais detalhados sobre a instalação de módulos de terceiros.

Iniciando um navegador controlado pelo Selenium

Para esses exemplos, será necessário ter o navegador web Firefox. Esse será o navegador que você controlará. Se você ainda não tem o Firefox, ele poderá ser baixado gratuitamente a partir de *http://getfirefox.com/*.

Importar os módulos para o Selenium é um pouco complicado. Em vez de usar import selenium, execute from selenium import webdriver. (O motivo exato pelo qual o módulo selenium está configurado dessa maneira está além do escopo deste livro.) Depois disso, o navegador Firefox poderá ser iniciado com o Selenium. Digite o seguinte no shell interativo:

```
>>> from selenium import webdriver
>>> browser = webdriver.Firefox()
>>> type(browser)
<class 'selenium.webdriver.firefox.webdriver.WebDriver'>
>>> browser.get('http://inventwithpython.com')
```

Você perceberá que, quando webdriver.Firefox() é chamado, o navegador web Firefox é iniciado. Chamar type() com o valor de webdriver.Firefox() mostra que ele é do tipo WebDriver. Além disso, chamar

browser.get('http://inventwithpython.com') direciona o navegador para *http://inventwithpython.com/*. Seu navegador deverá ter uma aparência semelhante à mostrada na figura 11.7.

Figura 11.7 – Após chamar webdriver.Firefox() *e* get() *no IDLE, o navegador Firefox aparecerá.*

Localizando elementos na página

Os objetos WebDriver têm alguns métodos para localizar elementos em uma página. Eles são divididos em métodos find_element_* e find_elements_*. Os métodos find_element_* retornam um único objeto WebElement que representa o primeiro elemento da página que corresponda à sua consulta. Os métodos find_elements_* retornam uma lista de objetos WebElement_* contendo *todos* os elementos correspondentes da página.

A tabela 11.3 mostra diversos exemplos de métodos find_element_* e find_elements_* chamados em um objeto WebDriver armazenado na variável browser.

Tabela 11.3 – Os métodos de WebDriver *do Selenium para encontrar elementos*

Nome do método	Objeto WebElement/lista retornados
browser.find_element_by_class_name(*nome*) browser.find_elements_by_class_name(*nome*)	Elementos que utilizam a classe CSS *nome*.
browser.find_element_by_css_selector(*seletor*) browser.find_elements_by_css_selector(*seletor*)	Elementos que correspondem ao *seletor* CSS.
browser.find_element_by_id(*id*)	Elementos com um valor de atributo

browser.find_elements_by_id(*id*)	*id* correspondente.
browser.find_element_by_link_text(*texto*) browser.find_elements_by_link_text(*texto*)	Elementos <a> que correspondem totalmente ao *texto* especificado.
browser.find_element_by_partial_link_text(*texto*) browser.find_elements_by_partial_link_text(*texto*)	Elementos <a> que contêm o *texto* especificado.
browser.find_element_by_name(*nome*) browser.find_elements_by_name(*nome*)	Elementos com um valor de atributo *nome* correspondente.
browser.find_element_by_tag_name(*nome*) browser.find_elements_by_tag_name(*nome*)	Elementos com uma tag *nome* correspondente (sem diferenciar letras maiúsculas de minúsculas; um elemento <a> corresponde a 'a' e a 'A').

Exceto pelos métodos *_by_tag_name(), os argumentos para todos os métodos consideram as diferenças entre letras maiúsculas e minúsculas. Se não houver nenhum elemento na página que corresponda ao que o método está procurando, o módulo selenium gerará uma exceção NoSuchElement. Se não quiser que essa exceção provoque uma falha em seu programa, acrescente instruções try e except em seu código.

Após ter o objeto WebElement, você poderá descobrir mais sobre ele ao ler os atributos ou chamar os métodos da tabela 11.4.

Tabela 11.4 – Atributos e métodos de WebElement

Atributo ou método	Descrição
tag_name	O nome da tag, por exemplo, 'a' para um elemento <a>.
get_attribute(*nome*)	O valor do atributo *nome* do elemento.
text	O texto do elemento, por exemplo, 'hello' em hello.
clear()	Para campos de texto ou elementos correspondentes à área de texto, limpa o texto digitado.
is_displayed()	Retorna True se o elemento estiver visível; caso contrário, retorna False.
is_enabled()	Para elementos de entrada, retorna True se o elemento estiver habilitado; caso contrário, retorna False.
is_selected()	Para elementos relacionados a caixas de seleção ou botões de rádio, retorna True se o elemento estiver selecionado; caso contrário, retorna False.

location	Um dicionário com chaves 'x' e 'y' para a posição do elemento na página.

Por exemplo, abra uma nova janela no editor de arquivo e digite o código a seguir:

```
from selenium import webdriver
browser = webdriver.Firefox()
browser.get('http://inventwithpython.com')
try:
    elem = browser.find_element_by_class_name('bookcover')
    print('Found <%s> element with that class name!' % (elem.tag_name))
except:
    print('Was not able to find an element with that name.')
```

Nesse caso, abrimos o Firefox e o direcionamos a um URL. Nessa página, tentamos encontrar elementos com nome de classe igual a 'bookcover' e, caso esse elemento seja encontrado, exibimos seu nome de tag usando o atributo tag_name. Se um elemento desse tipo não for encontrado, exibimos uma mensagem diferente.

Esse programa apresentará a saída a seguir:

```
Found <img> element with that class name!
```

Encontramos um elemento com o nome de classe 'bookcover' e o nome da tag 'img'.

Clicando na página

Os objetos WebElement retornados pelos métodos find_element_* e find_elements_* têm um método click() que simula um clique de mouse nesse elemento. Esse método pode ser usado para seguir um link, fazer uma seleção em um botão de rádio, clicar em um botão Submit (Submeter) ou disparar qualquer evento que possa ocorrer quando esse elemento for clicado pelo mouse. Por exemplo, digite o seguinte no shell interativo:

```
>>> from selenium import webdriver
>>> browser = webdriver.Firefox()
>>> browser.get('http://inventwithpython.com')
>>> linkElem = browser.find_element_by_link_text('Read It Online')
>>> type(linkElem)
```

```
<class 'selenium.webdriver.remote.webelement.WebElement'>
>>> linkElem.click()    # segue o link "Read It Online"
```

Esses comandos abrem o Firefox em *http://inventwithpython.com/*, obtêm o objeto WebElement para o elemento <a> com o texto *Read It Online* e então simulam um clique nesse elemento <a>. É como se você mesmo tivesse clicado no link; o navegador então seguirá esse link.

Preenchendo e submetendo formulários

Enviar teclas aos campos de texto em uma página web é uma questão de encontrar o elemento <input> ou <textarea> para esse campo de texto e chamar o método send_keys(). Por exemplo, digite o seguinte no shell interativo:

```
>>> from selenium import webdriver
>>> browser = webdriver.Firefox()
>>> browser.get('https://mail.yahoo.com')
>>> emailElem = browser.find_element_by_id('login-username')
>>> emailElem.send_keys('not_my_real_email')
>>> passwordElem = browser.find_element_by_id('login-passwd')
>>> passwordElem.send_keys('12345')
>>> passwordElem.submit()
```

Se o Gmail não alterou o id dos campos de texto Username (Usuário) e Password (Senha) desde que este livro foi publicado, o código anterior preencherá esses campos de texto com o texto fornecido. (Você sempre poderá usar o inspetor do navegador para conferir o id.) Chamar o método submit() em qualquer elemento produzirá o mesmo resultado que clicar no botão Submit do formulário em que o elemento estiver. (Você poderia igualmente ter chamado emailElem.submit(), e o código teria feito o mesmo.)

Enviando teclas especiais

O Selenium tem um módulo para teclas que são impossíveis de digitar em um valor de string que funciona de modo muito semelhante aos caracteres de escape. Esses valores são armazenados em atributos no módulo selenium.webdriver.common.keys. Como esse é um nome de módulo bem extenso, será muito mais fácil executar from selenium.webdriver.common.keys import Keys no início de seu programa; se fizer isso, você poderá

simplesmente escrever Keys em qualquer lugar que devesse normalmente escrever selenium.webdriver.common.keys. A tabela 11.5 lista as variáveis comumente utilizadas de Keys.

Tabela 11.5 – Variáveis comumente utilizadas do módulo selenium.webdriver.common.keys

Atributos	Significados
Keys.DOWN, Keys.UP, Keys.LEFT, Keys.RIGHT	As teclas de direção do teclado.
Keys.ENTER, Keys.RETURN	As teclas ENTER e RETURN.
Keys.HOME, Keys.END, Keys.PAGE_DOWN, Keys.PAGE_UP	As teclas HOME, END, PAGEDOWN e PAGEUP.
Keys.ESCAPE, Keys.BACK_SPACE, Keys.DELETE	As teclas ESC, BACKSPACE e DELETE.
Keys.F1, Keys.F2, . . . , Keys.F12	As teclas de F1 a F12 na parte superior do teclado.
Keys.TAB	A tecla TAB.

Por exemplo, se o cursor não estiver em um campo de texto no momento, pressionar as teclas HOME e END fará o navegador fazer rolagens para o início e o fim da página, respectivamente. Digite o seguinte no shell interativo e observe como as chamadas a send_keys() fazem rolagens na página:

```
>>> from selenium import webdriver
>>> from selenium.webdriver.common.keys import Keys
>>> browser = webdriver.Firefox()
>>> browser.get('http://nostarch.com')
>>> htmlElem = browser.find_element_by_tag_name('html')
>>> htmlElem.send_keys(Keys.END)     # faz rolagens para o final
>>> htmlElem.send_keys(Keys.HOME)    # faz rolagens para o início
```

A tag <html> é a tag básica dos arquivos HTML: o conteúdo completo do arquivo HTML é inserido entre tags <html> e </html>. Chamar browser.find_element_by_tag_name('html') é um bom lugar para enviar teclas para a página web em geral. Isso será útil, por exemplo, se um novo conteúdo for carregado depois que você tiver feito rolagens para o final da página.

Clicando nos botões do navegador

O Selenium também é capaz de simular cliques em diversos botões do navegador por meio dos métodos a seguir:

browser.back() Clica no botão Back (Retornar).

browser.forward() Clica no botão Forward (Avançar).

browser.refresh() Clica no botão Refresh/Reload (Atualizar/Recarregar).

browser.quit() Clica no botão Close Window (Fechar janela).

Mais informações sobre o Selenium

O Selenium pode fazer muito mais do que as funções descritas aqui. Ele pode modificar os cookies de seu navegador, capturar imagens de tela das páginas web e executar JavaScript personalizado. Para saber mais sobre esses recursos, acesse a documentação do Selenium em *http://selenium-python.readthedocs.org/*.

Resumo

A maioria das tarefas maçantes não está limitada aos arquivos em seu computador. Ser capaz de fazer download de páginas web por meio de programação estenderá seus programas à Internet. O módulo requests simplifica os downloads e, com um pouco de conhecimento básico dos conceitos de HTML e de seletores, você poderá utilizar o módulo BeautifulSoup para fazer parse das páginas baixadas.

No entanto, para automatizar completamente qualquer tarefa baseada em web, será necessário ter controle direto de seu navegador web por meio do módulo selenium. O módulo selenium permite fazer login em sites e preencher formulários automaticamente. Como um navegador web é a maneira mais comum de enviar e receber informações pela Internet, essa é uma ótima habilidade para ter em seu kit de ferramentas de programador.

Exercícios práticos

1. Descreva brevemente as diferenças entre os módulos webbrowser, requests, BeautifulSoup e selenium.

2. Que tipo de objeto é retornado por requests.get()? Como podemos acessar o conteúdo baixado na forma de um valor de string?
3. Que método de Requests verifica se o download funcionou?
4. Como o código de status HTTP de uma resposta de Requests pode ser obtido?
5. Como podemos salvar uma resposta de Requests em um arquivo?
6. Qual é o atalho de teclado para abrir as ferramentas de desenvolvedor em um navegador?
7. Como podemos visualizar o HTML (nas ferramentas de desenvolvedor) de um elemento específico de uma página web?
8. Qual é a string de seletor CSS que encontra o elemento com um atributo id igual a main?
9. Qual é a string de seletor CSS que encontra os elementos com uma classe CSS igual a highlight?
10. Qual é a string de seletor CSS que encontra todos os elementos <div> em outro elemento <div>?
11. Qual é a string de seletor CSS que encontra o elemento <button> com um atributo value definido com favorite?
12. Suponha que você tenha um objeto Tag de Beautiful Soup armazenado na variável spam para o elemento <div>Hello world!</div>. Como podemos obter a string 'Hello world!' do objeto Tag?
13. Como podemos armazenar todos os atributos de um objeto Tag de Beautiful Soup em uma variável chamada linkElem?
14. Executar import selenium não funciona. Como podemos importar devidamente o módulo selenium?
15. Qual é a diferença entre os métodos find_element_* e os métodos find_elements_*?
16. Quais métodos os objetos WebElement de Selenium têm para simular cliques de mouse e teclas?
17. Você poderia chamar send_keys(Keys.ENTER) no objeto WebElement do botão Submit, porém qual é a maneira mais fácil de submeter um formulário usando o Selenium?

18. Como podemos simular os cliques nos botões Forward (Avançar), Back (Retornar) e Refresh (Atualizar) de um navegador com o Selenium?

Projetos práticos

Para exercitar, escreva programas que executem as tarefas a seguir.

Envio de emails pela linha de comando

Crie um programa que aceite um endereço de email e uma string de texto na linha de comando e então, usando o Selenium, faça login em sua conta de email e envie um email contendo a string para o endereço fornecido. (Você poderá criar uma conta separada de email para esse programa.)

Essa seria uma maneira interessante de acrescentar uma funcionalidade de notificação em seus programas. Você também pode criar um programa semelhante para enviar mensagens de uma conta do Facebook ou do Twitter.

Programa para fazer downloads de um site de imagens

Crie um programa que acesse um site de compartilhamento de fotos, por exemplo, o Flickr ou o Imgur, pesquise uma classe de fotos e faça download de todas as imagens resultantes. Você pode criar um programa que funcione com qualquer site de fotos com um recurso de pesquisa.

2048

O 2048 é um jogo simples em que você combina blocos deslizando-os para cima, para baixo, para a esquerda e para a direita usando as teclas de direção. Você pode obter uma pontuação bem alta ao deslizar repetidamente um padrão para cima, para a direita, para baixo e para a esquerda continuamente. Crie um programa que abra o jogo em *https://gabrielecirulli.github.io/2048/* e fique enviando teclas para cima, para a direita, para baixo e para a esquerda para jogar automaticamente.

Verificação de links

Crie um programa que, dado o URL de uma página web, tentará fazer

download de todas as páginas com links nessa página. O programa deverá informar qualquer página que tenha um código de status 404 “Not Found” (Não encontrado) e exibi-las como links indisponíveis.

---

1 A resposta é não.

CAPÍTULO 12

TRABALHANDO COM PLANILHAS EXCEL

O Excel é uma aplicação popular e eficaz de planilhas para Windows. O módulo openpyxl permite que seus programas Python leiam e modifiquem arquivos de planilhas Excel. Por exemplo, talvez você tenha a tarefa maçante de copiar determinados dados de uma planilha

para outra. Quem sabe, você precise verificar milhares de linhas e selecionar apenas algumas delas para fazer pequenas alterações de acordo com alguns critérios. Pode ser que você precise olhar centenas de planilhas de orçamentos para o departamento e identificar aquelas que estejam no vermelho. São exatamente esses tipos de tarefas maçantes e automáticas que o Python poderá fazer para você.

Embora o Excel seja um software proprietário da Microsoft, há alternativas gratuitas que executam no Windows, no OS X e no Linux. Tanto o LibreOffice Calc quanto o OpenOffice Calc trabalham com o formato de arquivo *.xlsx* das planilhas Excel, o que significa que o módulo openpyxl poderá lidar também com planilhas desses aplicativos. Você pode fazer o download desses softwares a partir de *https://www.libreoffice.org/* e de *http://www.openoffice.org/*, respectivamente. Mesmo que o Excel já esteja instalado em seu computador, você poderá achar esses programas mais simples de usar. As capturas de tela deste capítulo, porém, foram todas feitas a partir do Excel 2010 no Windows 7.

Documentos Excel

Inicialmente, vamos ver algumas definições básicas: um documento de planilha Excel é chamado de *workbook*. Um único workbook é salvado em um arquivo com extensão *.xlsx*. Cada workbook contém várias *planilhas* (sheets, também chamadas de *worksheets*). A planilha que está sendo visualizada no momento (ou a última visualizada antes de fechar o Excel) se chama *planilha ativa* (active sheet).

Cada planilha contém *colunas* (acessadas por meio de letras que começam

com *A*) e *linhas* (acessadas por meio de números que começam com 1). Uma caixa em uma coluna e linha em particular é chamada de *célula* (cell). Cada célula pode conter um valor numérico ou um texto. A grade de células com dados forma uma planilha.

Instalando o módulo openpyxl

O Python não vem com o OpenPyXL, portanto será necessário instalá-lo. Siga as instruções para instalação de módulos de terceiros no Apêndice A; o nome do módulo é openpyxl. Para testar se esse módulo foi instalado corretamente, digite o seguinte no shell interativo:

```
>>> import openpyxl
```

Se o módulo for instalado corretamente, essa instrução não deverá gerar nenhuma mensagem de erro. Lembre-se de importar o módulo openpyxl antes de executar os exemplos deste capítulo no shell interativo; caso contrário, você obterá um erro NameError: name 'openpyxl' is not defined (NameError: o nome 'openpyxl' não está definido).

Este livro inclui a versão 2.1.4 do OpenPyXL, porém novas versões são regularmente disponibilizadas pela equipe que trabalha com esse módulo. Mas não se preocupe: as instruções usadas neste livro nas novas versões deverão ser compatíveis com as versões anteriores durante um bom tempo. Se você tiver uma versão mais recente e quiser ver quais recursos adicionais estão disponíveis a você, consulte a documentação completa do OpenPyXL em *http://openpyxl.readthedocs.org/*.

Lendo documentos Excel

Os exemplos deste capítulo usarão uma planilha chamada *example.xlsx* armazenada na pasta-raiz. Você pode criar a planilha por conta própria ou fazer o seu download a partir de *http://nostarch.com/automatestuff/*. A figura 12.1 mostra as abas das três planilhas default chamadas *Sheet1*, *Sheet2* e *Sheet3* que o Excel disponibiliza automaticamente para novos workbooks. (O número de planilhas default criadas pode variar de acordo com os sistemas operacionais e os programas de planilhas.)

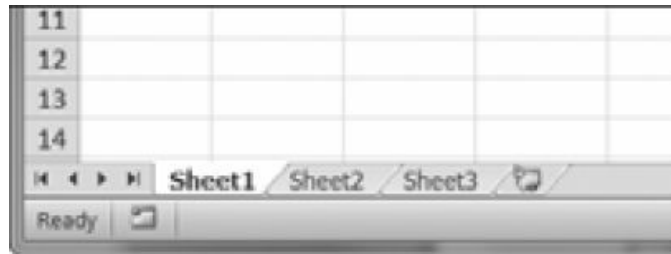



Figura 12.1 – As abas das planilhas de um workbook ficam no canto inferior esquerdo do Excel.

No arquivo de exemplo, Sheet 1 deverá ter a aparência da tabela 12.1. (Se você não fez o download de *example.xlsx* do site, digite esses dados na planilha por conta própria.)

Tabela 12.1 – A planilha example.xlsx

	A	B	C
1	4/5/2015 1:34:02 PM	Apples	73
2	4/5/2015 3:41:23 AM	Cherries	85
3	4/6/2015 12:46:51 PM	Pears	14
4	4/8/2015 8:59:43 AM	Oranges	52
5	4/10/2015 2:07:00 AM	Apples	152
6	4/10/2015 6:10:37 PM	Bananas	23
7	4/10/2015 2:40:46 AM	Strawberries	98

Agora que temos nossa planilha de exemplo, vamos ver como podemos manipulá-la usando o módulo openpyxl.

Abrindo documentos Excel com o OpenPyXL

Após ter importado o módulo openpyxl, a função openpyxl.load_workbook() poderá ser utilizada. Digite o seguinte no shell interativo:

```
>>> import openpyxl
>>> wb = openpyxl.load_workbook('example.xlsx')
>>> type(wb)
<class 'openpyxl.workbook.workbook.Workbook'>
```

A função openpyxl.load_workbook() recebe o nome do arquivo e retorna um valor com o tipo de dado workbook. Esse objeto Workbook representa o arquivo Excel, de modo um pouco semelhante à maneira como um objeto File

representa um arquivo-texto aberto.

Lembre-se de que *example.xlsx* deve estar no diretório de trabalho atual para que você possa trabalhar com ele. Você pode descobrir qual é o diretório de trabalho atual importando o módulo os e usando os.getcwd(); além disso, o diretório de trabalho atual poderá ser alterado com os.chdir().

Obtendo as planilhas do workbook

Uma lista com os nomes de todas as planilhas do workbook pode ser obtida por meio da chamada ao método get_sheet_names(). Digite o seguinte no shell interativo:

```
>>> import openpyxl
>>> wb = openpyxl.load_workbook('example.xlsx')
>>> wb.get_sheet_names()
['Sheet1', 'Sheet2', 'Sheet3']
>>> sheet = wb.get_sheet_by_name('Sheet3')
>>> sheet
<Worksheet "Sheet3">
>>> type(sheet)
<class 'openpyxl.worksheet.worksheet.Worksheet'>
>>> sheet.title
'Sheet3'
>>> anotherSheet = wb.get_active_sheet()
>>> anotherSheet
<Worksheet "Sheet1">
```

Cada planilha é representada por um objeto Worksheet, que pode ser obtido se passarmos a string com o nome da planilha para o método get_sheet_by_name() do workbook. Por fim, podemos chamar o método get_active_sheet() de um objeto Workbook para obter a planilha ativa do workbook. A planilha ativa é aquela que estará em evidência quando o workbook for aberto no Excel. De posse do objeto Worksheet, você poderá obter o seu nome a partir do atributo title.

Obtendo as células das planilhas

Depois que tiver um objeto Worksheet, um objeto Cell poderá ser acessado pelo seu nome. Digite o seguinte no shell interativo:

```
>>> import openpyxl
>>> wb = openpyxl.load_workbook('example.xlsx')
>>> sheet = wb.get_sheet_by_name('Sheet1')
>>> sheet['A1']
<Cell Sheet1.A1>
>>> sheet['A1'].value
datetime.datetime(2015, 4, 5, 13, 34, 2)
>>> c = sheet['B1']
>>> c.value
'Apples'
>>> 'Row ' + str(c.row) + ', Column ' + c.column + ' is ' + c.value
'Row 1, Column B is Apples'
>>> 'Cell ' + c.coordinate + ' is ' + c.value
'Cell B1 is Apples'
>>> sheet['C1'].value
73
```

O objeto Cell tem um atributo value que contém o valor armazenado nessa célula. Os objetos Cell também têm atributos row, column e coordinate que fornecem informações sobre a localização da célula.

Nesse caso, acessar o atributo value de nosso objeto Cell para a célula B1 nos dá a string 'Apples'. O atributo row nos fornece o inteiro 1, o atributo column nos fornece 'B' e o atributo coordinate nos dá 'B1'.

O OpenPyXL interpretará automaticamente as datas na coluna A e as retornará como valores datetime em vez de strings. O tipo de dado datetime será explicado com mais detalhes no capítulo 16.

Especificar uma coluna pela letra pode ser complicado para programar, especialmente porque, após a coluna Z, as colunas começam a usar duas letras: AA, AB, AC e assim por diante. Como alternativa, podemos também obter uma célula usando o método cell() da planilha e passando inteiros para seus argumentos nomeados row e column. O inteiro correspondente à primeira linha ou coluna é 1, e não 0. Prossiga com o exemplo no shell interativo digitando o seguinte:

```
>>> sheet.cell(row=1, column=2)
<Cell Sheet1.B1>
>>> sheet.cell(row=1, column=2).value
'Apples'
```

```
>>> for i in range(1, 8, 2):
        print(i, sheet.cell(row=i, column=2).value)

1 Apples
3 Pears
5 Apples
7 Strawberries
```

Como podemos ver, usar o método cell() da planilha e passar row=1 e column=2 fará você obter um objeto Cell para a célula B1, assim como ocorreu quando especificamos sheet['B1']. Então, ao usar o método cell() e seus argumentos nomeados, podemos criar um loop for para exibir os valores de uma série de células.

Suponha que você queira percorrer a coluna B e exibir o valor de todas as células que tenham um número de linha ímpar. Ao passar 2 para o parâmetro de "incremento" da função range(), podemos obter as células a cada duas linhas (nesse caso, todas as linhas com números ímpares). A variável i do loop for é passada para o argumento nomeado row do método cell(), enquanto 2 é sempre passado para o argumento nomeado column. Observe que passamos o inteiro 2, e não a string 'B'.

Podemos determinar o tamanho da planilha usando os métodos get_highest_row() e get_highest_column() do objeto Worksheet. Digite o seguinte no shell interativo:

```
>>> import openpyxl
>>> wb = openpyxl.load_workbook('example.xlsx')
>>> sheet = wb.get_sheet_by_name('Sheet1')
>>> sheet.get_highest_row()
7
>>> sheet.get_highest_column()
3
```

Observe que o método get_highest_column() retorna um inteiro, e não a letra que aparece no Excel.

Fazendo a conversão entre letras e números das colunas

Para fazer a conversão de letras para números, chame a função

openpyxl.cell.column_index_from_string(). Para converter de números para letras, chame a função openpyxl.cell.get_column_letter(). Digite o seguinte no shell interativo:

```
>>> import openpyxl
>>> from openpyxl.cell import get_column_letter, column_index_from_string
>>> get_column_letter(1)
'A'
>>> get_column_letter(2)
'B'
>>> get_column_letter(27)
'AA'
>>> get_column_letter(900)
'AHP'
>>> wb = openpyxl.load_workbook('example.xlsx')
>>> sheet = wb.get_sheet_by_name('Sheet1')
>>> get_column_letter(sheet.get_highest_column())
'C'
>>> column_index_from_string('A')
1
>>> column_index_from_string('AA')
27
```

Após ter importado essas duas funções do módulo openpyxl.cell, podemos chamar a função get_column_letter() e passar-lhe um inteiro, por exemplo, 27, para descobrir qual é a letra da 27ª coluna. A função column_index_string() faz o inverso: ela recebe a letra referente a uma coluna e informa o número dessa coluna. Não é necessário ter um workbook carregado para usar essas funções. Se quiser, você poderá carregar um workbook, obter um objeto Worksheet e chamar um método desse objeto, por exemplo, get_highest_column(), para obter um inteiro. Então você poderá passar esse inteiro para get_column_letter().

Obtendo linhas e colunas das planilhas

Podemos gerar slices de objetos Worksheet para obter todos os objetos Cell de uma linha, uma coluna ou uma área retangular da planilha. Então você poderá percorrer todas as células desse slice em um loop. Digite o seguinte no shell interativo:

```
>>> import openpyxl
>>> wb = openpyxl.load_workbook('example.xlsx')
>>> sheet = wb.get_sheet_by_name('Sheet1')
>>> tuple(sheet['A1':'C3'])
((<Cell Sheet1.A1>, <Cell Sheet1.B1>, <Cell Sheet1.C1>), (<Cell Sheet1.A2>, <Cell
Sheet1.B2>, <Cell Sheet1.C2>), (<Cell Sheet1.A3>, <Cell Sheet1.B3>, <Cell
Sheet1.C3>))
u >>> for rowOfCellObjects in sheet['A1':'C3']:
v         for cellObj in rowOfCellObjects:
              print(cellObj.coordinate, cellObj.value)
          print('--- END OF ROW ---')

A1 2015-04-05 13:34:02
B1 Apples
C1 73
--- END OF ROW ---
A2 2015-04-05 03:41:23
B2 Cherries
C2 85
--- END OF ROW ---
A3 2015-04-06 12:46:51
B3 Pears
C3 14
--- END OF ROW ---
```

Nesse caso, definimos que queremos objetos Cell da área retangular de A1 a C3 e obtivemos um objeto Generator contendo os objetos Cell dessa área. Para nos ajudar a visualizar esse objeto Generator, podemos usar tuple() nesse objeto de modo a exibir seus objetos Cell em uma tupla.

Essa tupla contém três tuplas, uma para cada linha, da parte superior da área desejada até a parte inferior. Cada uma dessas três tuplas internas contém os objetos Cell de uma linha da área desejada, da célula mais à esquerda para a célula mais à direita. De modo geral, nosso slice da planilha contém todos os objetos Cell da área de A1 a C3, começando na célula superior à esquerda e terminando na célula inferior à direita.

Para exibir os valores de cada célula dessa área, utilizamos dois loops for. O loop for externo percorre todas as linhas do slice u. Então, para cada linha, o loop for aninhado percorre todas as células dessa linha v.

Para acessar os valores das células de uma linha ou de uma coluna em particular, também podemos usar os atributos rows e columns de um objeto Worksheet. Digite o seguinte no shell interativo:

```
>>> import openpyxl
>>> wb = openpyxl.load_workbook('example.xlsx')
>>> sheet = wb.get_active_sheet()
>>> sheet.columns[1]
(<Cell Sheet1.B1>, <Cell Sheet1.B2>, <Cell Sheet1.B3>, <Cell Sheet1.B4>,
<Cell Sheet1.B5>, <Cell Sheet1.B6>, <Cell Sheet1.B7>)
>>> for cellObj in sheet.columns[1]:
        print(cellObj.value)

Apples
Cherries
Pears
Oranges
Apples
Bananas
Strawberries
```

Ao usar o atributo rows de um objeto Worksheet, vamos obter uma tupla de tuplas. Cada uma dessas tuplas internas representa uma linha e contém os objetos Cell dessa linha. O atributo columns também fornece uma tupla de tuplas, em que cada uma das tuplas internas contém objetos Cell de uma coluna em particular. Em *example.xlsx*, como há 7 linhas e 3 colunas, rows fornece uma tupla de 7 tuplas (cada qual contendo 3 objetos Cell) e columns fornece uma tupla de 3 tuplas (cada qual contendo 7 objetos Cell).

Para acessar uma tupla em particular, podemos referenciá-la pelo seu índice na tupla maior. Por exemplo, para obter a tupla que representa a coluna B, utilize sheet.columns[1]. Para obter a tupla que contém os objetos Cell da coluna A, utilize sheet.columns[0]. Depois que você tiver uma tupla que represente uma linha ou uma coluna, será possível percorrer seus objetos Cell por meio de um loop e exibir seus valores.

Workbooks, planilhas e células

Para fazer uma revisão rápida, apresentamos um resumo de todas as funções,

dos métodos e tipos de dados envolvidos na leitura de uma célula de um arquivo de planilha:

1. Importar o módulo openpyxl.
2. Chamar a função openpyxl.load_workbook().
3. Obter um objeto Workbook.
4. Chamar o método de workbook get_active_sheet() ou get_sheet_by_name().
5. Obter um objeto Worksheet.
6. Usar a indexação ou o método de planilha cell() com os argumentos nomeados row e column.
7. Obter um objeto Cell.
8. Ler o atributo value do objeto Cell.

Projeto: Ler dados de uma planilha

Suponha que você tenha uma planilha com dados do censo norte-americano de 2010 e tenha a tarefa maçante de verificar seus milhares de linhas para contabilizar a população total e o número de setores censitários de cada condado. (Um setor censitário é simplesmente uma área geográfica definida para o censo.) Cada linha representa um único setor censitário. Chamaremos o arquivo com a planilha de *censuspopdata.xlsx* e ele poderá ser baixado de *http://nostarch.com/automatestuff/*. Seu conteúdo é semelhante ao mostrado na figura 12.2.

	A	B	C	D	E
1	CensusTract	State	County	POP2010	
9841	06075010500	CA	San Francisco	2685	
9842	06075010600	CA	San Francisco	3894	
9843	06075010700	CA	San Francisco	5592	
9844	06075010800	CA	San Francisco	4578	
9845	06075010900	CA	San Francisco	4320	
9846	06075011000	CA	San Francisco	4827	
9847	06075011100	CA	San Francisco	5164	

Population by Census Tract

Ready

Figura 12.2 – A planilha censuspopdata.xlsx.

Apesar de o Excel poder calcular a soma de várias células selecionadas, continuaria sendo necessário selecionar as células de cada um dos mais de 3

mil condados. Mesmo que demore apenas alguns segundos para calcular a população de um condado manualmente, demoraria horas para que isso fosse feito para toda a planilha.

Nesse projeto, criaremos um script que possa ler o arquivo da planilha de censo e fazer cálculos estatísticos para cada condado em segundos.

Eis o que o seu programa deve fazer:

- Ler os dados da planilha Excel.
- Contabilizar o número de setores censitários em cada condado.
- Contabilizar a população total de cada condado.
- Exibir os resultados.

Isso significa que seu código deverá fazer o seguinte:

- Abrir e ler as células de um documento Excel com o módulo openpyxl.
- Fazer os cálculos com os dados dos setores censitários e da população e armazenar os resultados em uma estrutura de dados.
- Gravar a estrutura de dados em um arquivo-texto com extensão *.py* usando o módulo pprint.

Passo 1: Ler os dados da planilha

Há apenas uma planilha chamada 'Population by Census Tract' em *censuspopdata.xlsx*, e cada linha armazena os dados de um único setor censitário. As colunas correspondem ao número do setor (A), à abreviação do estado (B), ao nome do condado (C) e à população do setor (D).

Abra uma nova janela no editor de arquivo e insira o código a seguir. Salve o arquivo como *readCensusExcel.py*.

```
  #! python3
  # readCensusExcel.py – Cria uma tabela com a população e o número de setores
  censitários
  # de cada condado.

u import openpyxl, pprint
  print('Opening workbook...')
v wb = openpyxl.load_workbook('censuspopdata.xlsx')
w sheet = wb.get_sheet_by_name('Population by Census Tract')
```

```
   countyData = {}

   # TODO: Preenche countyData com a população e os setores de cada condado.
   print('Reading rows...')
x  for row in range(2, sheet.get_highest_row() + 1):
       # Cada linha da planilha contém dados de um setor censitário.
       state  = sheet['B' + str(row)].value
       county = sheet['C' + str(row)].value
       pop    = sheet['D' + str(row)].value

   # TODO: Abre um novo arquivo-texto e grava o conteúdo de countyData nesse arquivo.
```

Esse código importa o módulo openpyxl, assim como o módulo pprint que será usado para exibir os dados finais do condado u. Em seguida, o código abre o arquivo *censuspopdata.xlsx* v, obtém a planilha com os dados do censo w e começa a fazer uma iteração pelas linhas x.

Observe que criamos também uma variável chamada countyData, que conterá as populações e a quantidade de setores contabilizados para cada condado. Antes de armazenar qualquer dado nessa variável, porém, será necessário determinar exatamente como os seus dados serão estruturados.

Passo 2: Preencher a estrutura de dados

A estrutura de dados armazenada em countyData será um dicionário com as abreviaturas dos estados como chaves. Cada abreviatura de estado será mapeada a outro dicionário cujas chaves serão strings com os nomes dos condados desse estado. Cada nome de condado, por sua vez, será mapeado a um dicionário que contém somente duas chaves: 'tracts' e 'pop'. Essas chaves são mapeadas para o número de setores censitários e a população do condado. Por exemplo, o dicionário será semelhante a:

```
{'AK': {'Aleutians East': {'pop': 3141, 'tracts': 1},
        'Aleutians West': {'pop': 5561, 'tracts': 2},
        'Anchorage': {'pop': 291826, 'tracts': 55},
        'Bethel': {'pop': 17013, 'tracts': 3},
        'Bristol Bay': {'pop': 997, 'tracts': 1},
        --trecho removido--
```

Se o dicionário anterior estiver armazenado em countyData, as expressões a

seguir serão avaliadas da seguinte maneira:

```
>>> countyData['AK']['Anchorage']['pop']
291826
>>> countyData['AK']['Anchorage']['tracts']
55
```

De modo geral, as chaves do dicionário countyData terão o seguinte aspecto:

countyData[*abreviatura do estado*][*condado*]['tracts']
countyData[*abreviatura do estado*][*condado*]['pop']

Agora que já sabemos como countyData será estruturado, podemos escrever o código que o preencherá com os dados dos condados. Acrescente o código a seguir no final de seu programa:

```
#! python 3
# readCensusExcel.py – Cria uma tabela com a população e o número de setores
censitários
# de cada condado.

--trecho removido--

for row in range(2, sheet.get_highest_row() + 1):
    # Cada linha da planilha contém dados de um setor censitário.
    state  = sheet['B' + str(row)].value
    county = sheet['C' + str(row)].value
    pop    = sheet['D' + str(row)].value

    # Garante que a chave para esse estado existe.
u   countyData.setdefault(state, {})
    # Garante que a chave para esse condado nesse estado existe.
v   countyData[state].setdefault(county, {'tracts': 0, 'pop': 0})

    # Cada linha representa um setor censitário, portanto incrementa o valor de um.
w   countyData[state][county]['tracts'] += 1
    # Soma a população desse setor censitário à população do condado.
x   countyData[state][county]['pop'] += int(pop)

# TODO: Abre um novo arquivo-texto e grava o conteúdo de countyData nesse arquivo.
```

As duas últimas linhas de código executam o trabalho de cálculo

propriamente dito, incrementando o valor de tracts w e somando um valor em pop x para o condado atual a cada iteração do loop for.

A outra parte do código está presente porque não podemos acrescentar o dicionário de um condado como o valor da chave referente a uma abreviatura de estado antes que essa chave esteja presente em countyData. (Isso quer dizer que countyData['AK']['Anchorage']['tracts'] += 1 provocará um erro se a chave 'AK' ainda não existir.) Para garantir que a chave com a abreviatura do estado exista em sua estrutura de dados, é preciso chamar o método setdefault() e definir um valor caso ainda não haja nenhum para state u.

Assim como o dicionário countyData precisa de um dicionário como valor para cada chave com a abreviatura de um estado, cada um *desses* dicionários precisará de seu próprio dicionário como valor para cada chave referente a um condado v. Cada um *desses* dicionários, por sua vez, precisará de chaves 'tracts' e 'pop' inicializadas com o valor inteiro 0. (Se você se confundir com a estrutura do dicionário, retorne ao dicionário de exemplo no início desta seção.)

Como setdefault() não fará nada se a chave já existir, ele poderá ser chamado em todas as iterações do loop for sem que haja problemas.

Passo 3: Gravar os resultados em um arquivo

Após o loop for ter sido concluído, o dicionário countyData conterá todas as informações de população e de setores censitários, acessíveis por chaves conforme o condado e o estado. A essa altura, você poderá implementar mais código para gravar esses dados em um arquivo-texto ou em outra planilha Excel. Por enquanto, vamos simplesmente utilizar a função pprint.pformat() para gravar o valor do dicionário countyData como uma string enorme em um arquivo chamado *census2010.py*. Acrescente o código a seguir no final de seu programa (garantindo que ele não esteja indentado para que permaneça fora do loop for):

```
#! python 3
# readCensusExcel.py – Cria uma tabela com a população e o número de setores
censitários
# de cada condado.
```

```
--trecho removido--

for row in range(2, sheet.get_highest_row() + 1):
--trecho removido--

# Abre um novo arquivo-texto e grava o conteúdo de countyData nesse arquivo.
print('Writing results...')
resultFile = open('census2010.py', 'w')
resultFile.write('allData = ' + pprint.pformat(countyData))
resultFile.close()
print('Done.')
```

A função pprint.pformat() gera uma string formatada como código Python válido. Ao gravá-la em um arquivo-texto chamado *census2010.py*, você terá gerado um programa Python com o seu programa Python! Isso pode parecer complicado, porém a vantagem é que, a partir de agora, será possível importar *census2010.py*, como qualquer outro módulo Python. No shell interativo, mude o diretório de trabalho atual para a pasta que contém seu arquivo *census2010.py* recém-criado (em meu laptop, é *C:\Python34*) e importe-o:

```
>>> import os
>>> os.chdir('C:\\Python34')
>>> import census2010
>>> census2010.allData['AK']['Anchorage']
{'pop': 291826, 'tracts': 55}
>>> anchoragePop = census2010.allData['AK']['Anchorage']['pop']
>>> print('The 2010 population of Anchorage was ' + str(anchoragePop))
The 2010 population of Anchorage was 291826
```

O programa *readCensusExcel.py* é um código descartável: depois que seus resultados forem salvados em *census2010.py*, não será necessário executar o programa novamente. Sempre que precisar dos dados dos condados, você poderá simplesmente executar import census2010.

Calcular esses dados manualmente exigiria horas; esse programa fez isso em alguns segundos. Ao usar o OpenPyXL, você não terá nenhum problema para extrair informações de uma planilha Excel e realizar cálculos com esse dados. O programa completo pode ser baixado a partir de *http://nostarch.com/automatestuff/*.

Ideias para programas semelhantes

Muitos negócios e escritórios usam o Excel para armazenar diversos tipos de dados, e não é incomum que as planilhas se tornem enormes e difíceis de manusear. Qualquer programa que faça parse em uma planilha Excel terá uma estrutura semelhante: o arquivo da planilha é carregado, algumas variáveis ou estruturas de dados são preparadas e as linhas da planilha são percorridas por meio de um loop. Um programa desse tipo pode fazer o seguinte:

- Comparar dados de diversas linhas em uma planilha.
- Abrir vários arquivos Excel e comparar dados entre as planilhas.
- Verificar se uma planilha contém linhas em branco ou dados inválidos em alguma célula e alertar o usuário caso isso ocorra.
- Ler dados de uma planilha e usá-los como entrada para seus programas Python.

Escrevendo em documentos Excel

O OpenPyXL também disponibiliza maneiras de escrever dados, o que quer dizer que seus programas poderão criar e editar arquivos de planilha. Em Python, é fácil criar planilhas com milhares de linhas de dados.

Criando e salvando documentos Excel

Chame a função `openpyxl.Workbook()` para criar um novo objeto `Workbook` vazio. Digite o seguinte no shell interativo:

```
>>> import openpyxl
>>> wb = openpyxl.Workbook()
>>> wb.get_sheet_names()
['Sheet']
>>> sheet = wb.get_active_sheet()
>>> sheet.title
'Sheet'
>>> sheet.title = 'Spam Bacon Eggs Sheet'
>>> wb.get_sheet_names()
['Spam Bacon Eggs Sheet']
```

O workbook será iniciado com uma única planilha chamada *Sheet*. O nome da planilha poderá ser alterado se uma nova string for armazenada em seu atributo title.

Sempre que o objeto Workbook ou suas planilhas e células forem modificados, o arquivo da planilha não será salvo até que o método save() do workbook seja chamado. Digite o seguinte no shell interativo (com *example.xlsx* no diretório de trabalho atual):

```
>>> import openpyxl
>>> wb = openpyxl.load_workbook('example.xlsx')
>>> sheet = wb.get_active_sheet()
>>> sheet.title = 'Spam Spam Spam'
>>> wb.save('example_copy.xlsx')
```

Nesse caso, alteramos o nome de nossa planilha. Para salvar nossas alterações, passamos um nome de arquivo como uma string ao método save(). Se um nome de arquivo diferente do original for passado, por exemplo, 'example_copy.xlsx', as alterações serão salvadas em uma cópia da planilha.

Sempre que editar uma planilha carregada a partir de um arquivo, salve a nova planilha editada com um nome de arquivo diferente do original. Dessa maneira, você ainda terá o arquivo original da planilha para trabalhar caso um bug em seu código faça com que o novo arquivo salvado tenha dados incorretos ou corrompidos.

Criando e removendo planilhas

As planilhas podem ser acrescentadas e removidas de um workbook com os métodos create_sheet() e remove_sheet(). Digite o seguinte no shell interativo:

```
>>> import openpyxl
>>> wb = openpyxl.Workbook()
>>> wb.get_sheet_names()
['Sheet']
>>> wb.create_sheet()
<Worksheet "Sheet1">
>>> wb.get_sheet_names()
['Sheet', 'Sheet1']
>>> wb.create_sheet(index=0, title='First Sheet')
```

```
<Worksheet "First Sheet">
>>> wb.get_sheet_names()
['First Sheet', 'Sheet', 'Sheet1']
>>> wb.create_sheet(index=2, title='Middle Sheet')
<Worksheet "Middle Sheet">
>>> wb.get_sheet_names()
['First Sheet', 'Sheet', 'Middle Sheet', 'Sheet1']
```

O método create_sheet() retorna um novo objeto Worksheet chamado Sheet*X*, que, por padrão, é definido como a última planilha do workbook. Opcionalmente, o índice e o nome da nova planilha podem ser especificados por meio dos argumentos nomeados index e title.

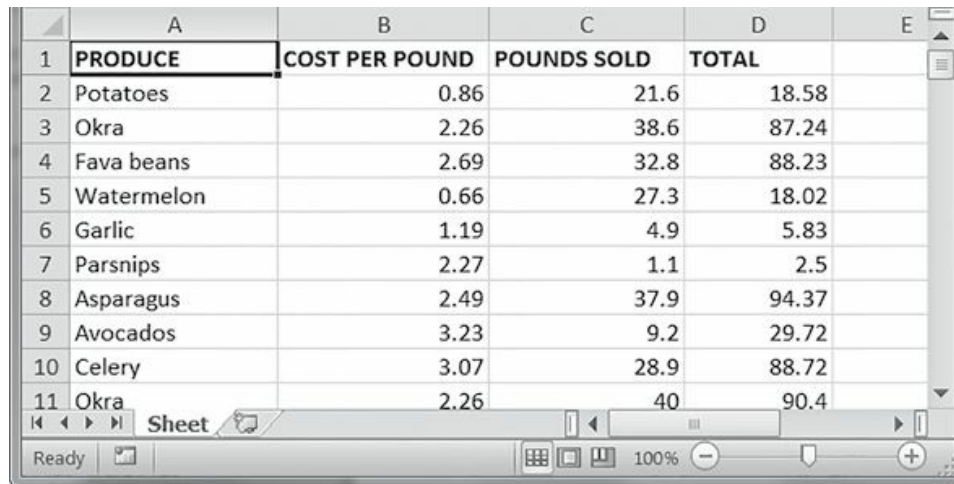
Prossiga com o exemplo anterior digitando o seguinte:

```
>>> wb.get_sheet_names()
['First Sheet', 'Sheet', 'Middle Sheet', 'Sheet1']
>>> wb.remove_sheet(wb.get_sheet_by_name('Middle Sheet'))
>>> wb.remove_sheet(wb.get_sheet_by_name('Sheet1'))
>>> wb.get_sheet_names()
['First Sheet', 'Sheet']
```

O método remove_sheet() aceita um objeto Worksheet como argumento, e não uma string com o nome da planilha. Se apenas o nome de uma planilha que você quer remover for conhecido, chame get_sheet_by_name() e passe seu valor de retorno para remove_sheet().

Lembre-se de chamar o método save() para salvar as alterações após ter adicionado ou removido planilhas do workbook.

Escrevendo valores em células

Escrever valores em células é muito semelhante a escrever valores para as chaves de um dicionário. Digite o seguinte no shell interativo:

```
>>> import openpyxl
>>> wb = openpyxl.Workbook()
>>> sheet = wb.get_sheet_by_name('Sheet')
>>> sheet['A1'] = 'Hello world!'
>>> sheet['A1'].value
'Hello world!'
```

Se você tiver a coordenada da célula como uma string, poderá usá-la como se fosse uma chave de dicionário no objeto Worksheet para especificar em qual célula a escrita será feita.

Projeto: Atualizando uma planilha

Nesse projeto, criaremos um programa para atualizar as células de uma planilha de venda de produtos. Seu programa percorrerá a planilha, encontrará tipos específicos de produtos e atualizará seus preços. Faça download dessa planilha a partir de *http://nostarch.com/automatestuff/*. A figura 12.3 mostra a aparência da planilha.

	A	B	C	D	E
1	PRODUCE	COST PER POUND	POUNDS SOLD	TOTAL	
2	Potatoes	0.86	21.6	18.58	
3	Okra	2.26	38.6	87.24	
4	Fava beans	2.69	32.8	88.23	
5	Watermelon	0.66	27.3	18.02	
6	Garlic	1.19	4.9	5.83	
7	Parsnips	2.27	1.1	2.5	
8	Asparagus	2.49	37.9	94.37	
9	Avocados	3.23	9.2	29.72	
10	Celery	3.07	28.9	88.72	
11	Okra	2.26	40	90.4	

Sheet

Ready 100%

Figura 12.3 – Uma planilha para venda de produtos.

Cada linha representa uma venda individual. As colunas contêm o tipo de produto vendido (A), o custo por libra (pound) desse produto (B), o número de libras vendido (C) e a receita total dessa venda (D). A coluna TOTAL está definida com a fórmula Excel *=ROUND(B3*C3, 2)*, que multiplica o custo por libra pelo número de libras vendido e arredonda o resultado para o centavo mais próximo. Com essa fórmula, as células na coluna TOTAL serão automaticamente atualizadas se houver uma mudança nas colunas B ou C.

Agora suponha que os preços do alho (garlic), do aipo (celery) e dos limões (lemons) tenham sido inseridos incorretamente, deixando você com a tarefa maçante de verificar milhares de linhas dessa planilha e atualizar o custo por libra para todas as linhas contendo alho, aipo e limão. Não é possível simplesmente pesquisar e substituir o preço, pois pode haver outros itens com

o mesmo preço que não devem ser erroneamente “corrigidos”. Se houver milhares de linhas, essa tarefa demorará horas para ser feita manualmente. No entanto, podemos criar um programa que faça isso em segundos.

Seu programa deve fazer o seguinte:

- Percorrer todas as linhas em um loop.
- Se a linha contiver alho (garlic), aipo (celery) ou limões (lemons), o preço deverá ser alterado.

Isso significa que seu código deverá fazer o seguinte:

- Abrir o arquivo com a planilha.
- Para cada linha, verificar se o valor na coluna A é Celery, Garlic ou Lemon.
- Em caso afirmativo, o preço na coluna B deverá ser atualizado.
- Salvar a planilha em um novo arquivo (para que você não perca a planilha antiga, somente por garantia).

Passo 1: Definir uma estrutura de dados com as informações a serem atualizadas

Os preços que devem ser atualizados são:

Aipo (celery) 1,19

Alho (garlic) 3,07

Limão (lemon) 1,27

Você pode escrever um código como este:

```
if produceName == 'Celery':
    cellObj = 1.19
if produceName == 'Garlic':
    cellObj = 3.07
if produceName == 'Lemon':
    cellObj = 1.27
```

Ter os dados dos produtos e dos preços atualizados fixos no código dessa maneira não é muito elegante. Se houver necessidade de atualizar a planilha novamente com preços e produtos diferentes, será preciso alterar uma grande quantidade de código. Sempre que o código é alterado, corremos o risco de introduzir bugs.

Uma solução mais flexível consiste em armazenar as informações corrigidas de preços em um dicionário e escrever seu código de modo a usar essa estrutura de dados. Em uma nova janela do editor de arquivo, insira o código a seguir:

```
#! python3
# updateProduce.py – Corrige os preços em uma planilha de venda de produtos.

import openpyxl

wb = openpyxl.load_workbook('produceSales.xlsx')
sheet = wb.get_sheet_by_name('Sheet')

# Os tipos de produto e seus preços atualizados
PRICE_UPDATES = {'Garlic': 3.07,
                 'Celery': 1.19,
                 'Lemon': 1.27}

# TODO: Percorre as linhas em um loop e atualiza os preços.
```

Salve esse código como *updateProduce.py*. Se for necessário atualizar a planilha novamente, não será preciso atualizar nenhum código além do dicionário PRICE_UPDATES.

Passo 2: Verificar todas as linhas e atualizar os preços incorretos

A próxima parte do programa percorrerá todas as linhas da planilha em um loop. Acrescente o código a seguir no final de *updateProduce.py*:

```
#! python3
# updateProduce.py – Corrige os preços em uma planilha de venda de produtos.

--trecho removido--

# Percorre as linhas em um loop e atualiza os preços.
u for rowNum in range(2, sheet.get_highest_row()):    # pula a primeira linha
v     produceName = sheet.cell(row=rowNum, column=1).value
w     if produceName in PRICE_UPDATES:
          sheet.cell(row=rowNum, column=2).value = PRICE_UPDATES[produceName]
```

```
x wb.save('updatedProduceSales.xlsx')
```

Percorremos as linhas em um loop a partir da linha 2, pois a linha 1 contém apenas o cabeçalho u. A célula na coluna 1 (ou seja, na coluna A) será armazenada na variável produceName v. Se produceName estiver presente como uma chave no dicionário PRICE_UPDATES w, saberemos que essa é uma linha que deverá ter seu preço corrigido. O preço correto estará em PRICE_UPDATES[produceName].

Observe como o uso de PRICE_UPDATES deixa o código limpo. Somente uma instrução if, e não um código como if produceName == 'Garlic':, será necessária para todos os tipos de produto a serem atualizados. Como o código utiliza o dicionário PRICE_UPDATES em vez de fixar os nomes dos produtos e os custos atualizados no código do loop for, será necessário modificar somente esse dicionário, e não o código, se a planilha de venda de produtos precisar de alterações adicionais.

Após percorrer toda a planilha e fazer as alterações, o código salvará o objeto Workbook em *updatedProduceSales.xlsx* x. A planilha antiga não será sobrescrita caso haja um bug em seu programa e se a planilha atualizada estiver incorreta. Após verificar se a planilha atualizada está correta, a planilha antiga poderá ser apagada.

O download do código-fonte completo desse programa poderá ser feito a partir de *http://nostarch.com/automatestuff/*.

Ideias para programas semelhantes

Como muitos funcionários de escritório usam planilhas Excel o tempo todo, um programa que possa editar e gravar arquivos Excel automaticamente poderá ser muito útil. Um programa desse tipo pode fazer o seguinte:

- Ler dados de uma planilha e escrevê-los em partes de outras planilhas.
- Ler dados de sites, arquivos-texto ou do clipboard e gravá-los em uma planilha.
- “Limpar” dados de planilhas automaticamente. Por exemplo, poderíamos usar expressões regulares para ler vários formatos de números de telefone e editá-los em um formato único e padronizado.

Definindo o estilo de fonte das células

Estilizar determinadas células, linhas ou colunas pode ajudar a enfatizar áreas importantes de sua planilha. Na planilha de venda de produtos, por exemplo, seu programa poderia aplicar negrito às linhas com batata (potato), alho (garlic) e pastinaca (parsnip). Talvez você queira deixar todas as linhas em que o custo por libra seja maior que cinco dólares em itálico. Estilizar partes de uma planilha grande manualmente pode ser tedioso, porém seus programas poderão fazer isso instantaneamente.

Para personalizar os estilos das fontes nas células, é preciso importar as funções Font() and Style() do módulo openpyxl.styles.

```
from openpyxl.styles import Font, Style
```

Isso permite digitar Font() no lugar de openpyxl.styles.Font(). (Veja a seção “Importando módulos” para rever esse estilo de instrução import.)

A seguir, apresentamos um exemplo que cria um novo workbook e define a célula A1 para que tenha uma fonte de 24 pontos em itálico. Digite o seguinte no shell interativo:

```
  >>> import openpyxl
  >>> from openpyxl.styles import Font, Style
  >>> wb = openpyxl.Workbook()
  >>> sheet = wb.get_sheet_by_name('Sheet')
u >>> italic24Font = Font(size=24, italic=True)
v >>> styleObj = Style(font=italic24Font)
w >>> sheet['A1'].style = styleObj
  >>> sheet['A1'] = 'Hello world!'
  >>> wb.save('styled.xlsx')
```

O OpenPyXL representa a coleção de parâmetros de estilo de uma célula com um objeto Style, que é armazenado no atributo style do objeto Cell. O estilo de uma célula pode ser especificado ao definir o atributo style com o objeto Style.

Nesse exemplo, Font(size=24, italic=True) retorna um objeto Font, que é armazenado em italic24Font u. Os argumentos nomeados size e italic de Font() configuram os atributos de style do objeto Font. Esse objeto Font então é passado para a chamada Style(font=italic24Font), que retorna o valor

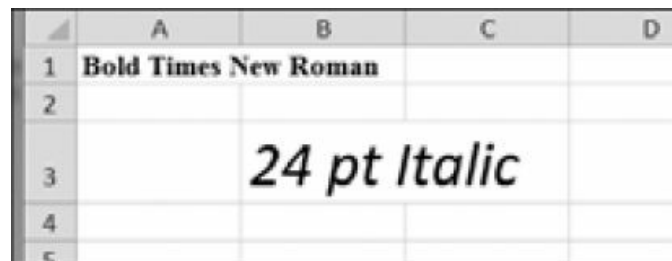
armazenado em styleObj v. Quando o atributo style da célula é definido com styleObj w, todas essas informações de estilização são aplicadas à célula A1.

Objetos Font

Os atributos de style em objetos Font afetam o modo como o texto é exibido nas células. Para definir os atributos de estilo da fonte, passe argumentos nomeados a Font(). A tabela 12.2 mostra os possíveis argumentos nomeados da função Font().

Tabela 12.2 – Argumentos nomeados para atributos de style *de fonte*

Argumento nomeado	Tipo de dado	Descrição
name	String	O nome da fonte, por exemplo, 'Calibri' ou 'Times New Roman'
size	Integer	O tamanho em pontos
bold	Boolean	True para fonte em negrito
italic	Boolean	True para fonte em itálico

Podemos chamar Font() para criar um objeto Font e armazenar esse objeto em uma variável. Passe esse objeto para Style(), armazene o objeto Style resultante em uma variável e defina o atributo style de um objeto Cell com essa variável. Por exemplo, o código a seguir cria diversos estilos de fonte:

```
>>> import openpyxl
>>> from openpyxl.styles import Font, Style
>>> wb = openpyxl.Workbook()
>>> sheet = wb.get_sheet_by_name('Sheet')

>>> fontObj1 = Font(name='Times New Roman', bold=True)
>>> styleObj1 = Style(font=fontObj1)
>>> sheet['A1'].style/styleObj
>>> sheet['A1'] = 'Bold Times New Roman'

>>> fontObj2 = Font(size=24, italic=True)
>>> styleObj2 = Style(font=fontObj2)
>>> sheet['B3'].style/styleObj
>>> sheet['B3'] = '24 pt Italic'
```

```
>>> wb.save('styles.xlsx')
```

Nesse caso, armazenamos um objeto Font em fontObj1 e o usamos para criar um objeto Style que armazenamos em styleObj1 e, em seguida, definimos o atributo style do objeto Cell de A1 com styleObj. Repetimos o processo com outro objeto Font e outro objeto Style para definir o estilo de uma segunda célula. Após executar esse código, os estilos das células A1 e B3 na planilha serão definidos com estilos personalizados de fonte, conforme mostrado na figura 12.4.

	A	B	C	D
1	Bold Times New Roman			
2				
3		24 pt Italic		
4				
5				

Figura 12.4 – Uma planilha com estilos personalizados de fonte.

Para a célula A1, definimos o nome da fonte com 'Times New Roman' e configuramos bold com True para que o texto apareça em Times New Roman em negrito. Não especificamos um tamanho, portanto o default de openpyxl, que é 11, é utilizado. Na célula B3, nosso texto está em itálico, com tamanho igual a 24; não especificamos um nome de fonte, portanto o default de openpyxl, que é Calibri, é usado.

Fórmulas

As fórmulas, que começam com um sinal de igualdade, podem configurar células para que contenham valores calculados a partir de outras células. Nesta seção, usaremos o módulo openpyxl para adicionar fórmulas em células por meio de programação, assim como fazemos com qualquer valor normal. Por exemplo:

```
>>> sheet['B9'] = '=SUM(B1:B8)'
```

Essa instrução armazena *=SUM(B1:B8)* como o valor da célula B9. A célula B9 é definida com uma fórmula que calcula a soma dos valores das células de B1 a B8. Você pode ver isso em ação na figura 12.5.

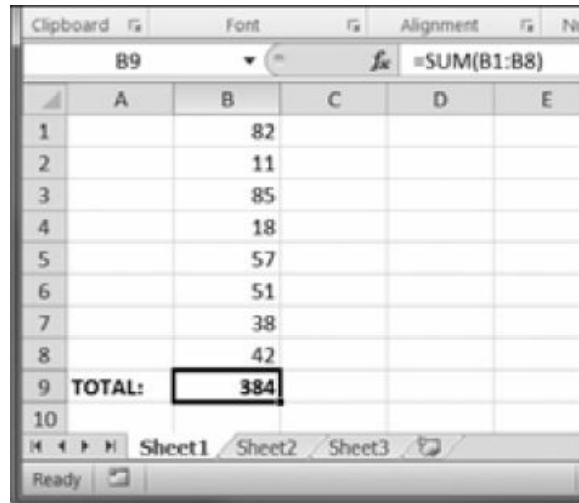



Figura 12.5 – A célula B9 contém a fórmula =SUM(B1:B8)*, que soma as células de B1 a B8.*

Uma fórmula é definida do mesmo modo que qualquer outro valor de texto em uma célula. Digite o seguinte no shell interativo:

```
>>> import openpyxl
>>> wb = openpyxl.Workbook()
>>> sheet = wb.get_active_sheet()
>>> sheet['A1'] = 200
>>> sheet['A2'] = 300
>>> sheet['A3'] = '=SUM(A1:A2)'
>>> wb.save('writeFormula.xlsx')
```

As células em A1 e em A2 são definidas com 200 e 300, respectivamente. O valor da célula A3 é definido com uma fórmula que soma os valores de A1 e de A2. Quando a planilha for aberta no Excel, A3 exibirá o valor 500.

A fórmula de uma célula também pode ser lida como seria feito com qualquer valor. No entanto, se você quiser ver o resultado do *cálculo* da fórmula, e não a fórmula literal, passe True para o argumento nomeado data_only de load_workbook(). Isso quer dizer que um objeto Workbook pode mostrar as fórmulas ou o resultado delas, mas não ambos. (Mas podemos ter vários objetos Workbook carregados para o mesmo arquivo de planilha.) Digite o seguinte no shell interativo para ver a diferença entre carregar um workbook com e sem o argumento nomeado data_only:

```
>>> import openpyxl
>>> wbFormulas = openpyxl.load_workbook('writeFormula.xlsx')
```

```
>>> sheet = wbFormulas.get_active_sheet()
>>> sheet['A3'].value
'=SUM(A1:A2)'

>>> wbDataOnly = openpyxl.load_workbook('writeFormula.xlsx', data_only=True)
>>> sheet = wbDataOnly.get_active_sheet()
>>> sheet['A3'].value
500
```

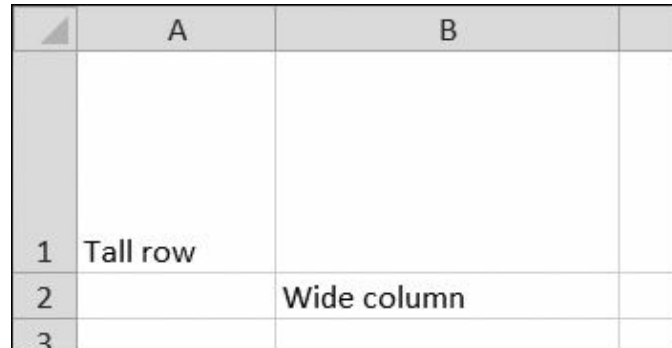
Nesse caso, quando load_workbook() é chamado com data_only=True, a célula A3 mostra 500, que é o resultado da fórmula *=SUM(A1:A2)*, e não o texto da fórmula.

As fórmulas em Excel permitem aplicar certo nível de programação em planilhas, mas podem rapidamente se tornar difíceis de administrar para tarefas complexas. Por exemplo, mesmo que você tenha bastante familiaridade com as fórmulas em Excel, tentar decifrar o que =IFERROR(TRIM(IF(LEN(VLOOKUP(F7, Sheet2!$A$1:$B$10000, 2, FALSE))>0,SUBSTITUTE(VLOOKUP(F7, Sheet2!$A$1:$B$10000, 2, FALSE), " ", ""),"")), "") realmente faz será uma dor de cabeça. O código Python é muito mais legível.

Ajustando linhas e colunas

Em Excel, ajustar os tamanhos das linhas e das colunas é muito simples: basta clicar e arrastar as bordas de um cabeçalho de linha ou de coluna. Porém, se for necessário definir o tamanho de uma linha ou de uma coluna de acordo com o conteúdo de suas células ou se você quiser definir os tamanhos em uma grande quantidade de arquivos de planilha, será muito mais rápido criar um programa Python que faça isso.

As linhas e colunas também podem ser totalmente ocultas. Além disso, elas podem ser “congeladas” de modo que permaneçam sempre visíveis na tela e apareçam em todas as páginas quando a planilha for exibida (o que é conveniente para cabeçalhos).

Definido a altura da linha e a largura da coluna

Os objetos Worksheet têm atributos row_dimensions e column_dimensions que

controlam as alturas das linhas e as larguras das colunas. Digite o seguinte no shell interativo:

```
>>> import openpyxl
>>> wb = openpyxl.Workbook()
>>> sheet = wb.get_active_sheet()
>>> sheet['A1'] = 'Tall row'
>>> sheet['B2'] = 'Wide column'
>>> sheet.row_dimensions[1].height = 70
>>> sheet.column_dimensions['B'].width = 20
>>> wb.save('dimensions.xlsx')
```

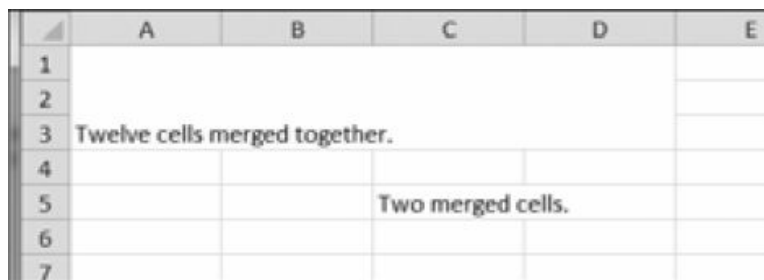
Os atributos row_dimensions e column_dimensions de uma planilha são valores semelhantes a dicionários; row_dimensions contém objetos RowDimension, e column_dimensions contém objetos ColumnDimension. Em row_dimensions, podemos acessar um dos objetos usando o número da linha (nesse caso, 1 ou 2). Em column_dimensions, podemos acessar um dos objetos usando a letra da coluna (nesse caso, A ou B).

A planilha *dimensions.xlsx* tem o aspecto mostrado na figura 12.6.

	A	B	
1	Tall row		
2		Wide column	
3			

Figura 12.6 – A linha 1 e a coluna B definidas com alturas e larguras maiores.

De posse do objeto RowDimension, você poderá definir a sua altura. Depois que tiver o objeto ColumnDimension, você poderá definir a sua largura. A altura da linha pode ser definida com um inteiro ou com um valor de ponto flutuante entre 0 e 409. Esse valor representa a altura medida em *pontos*, em que um ponto é igual a 1/72 de uma polegada. A altura default de uma linha é 12,75. A largura da coluna pode ser definida com um inteiro ou com um valor de ponto flutuante entre 0 e 255. Esse valor representa o número de caracteres com o tamanho default de fonte (11 pontos) que pode ser exibido

na célula. A largura default da coluna corresponde a 8,43 caracteres. As colunas com largura igual a 0 ou as linhas com altura igual a 0 permanecerão ocultas ao usuário.

Mesclar e separar células

Uma área retangular de células pode ser mesclada e formar uma única célula com o método de planilha merge_cells(). Digite o seguinte no shell interativo:

```
>>> import openpyxl
>>> wb = openpyxl.Workbook()
>>> sheet = wb.get_active_sheet()
>>> sheet.merge_cells('A1:D3')
>>> sheet['A1'] = 'Twelve cells merged together.'
>>> sheet.merge_cells('C5:D5')
>>> sheet['C5'] = 'Two merged cells.'
>>> wb.save('merged.xlsx')
```

O argumento de merge_cells() é uma única string com as células do canto superior esquerdo e do canto inferior direito da área retangular a ser mesclada: 'A1:D3' combina doze células em uma única célula. Para definir o valor dessas células mescladas, basta especificar o valor da célula superior esquerda do grupo resultante da combinação.

Ao executar esse código, *merged.xlsx* terá a aparência mostrada na figura 12.7.

Figura 12.7 – Células mescladas em uma planilha.

Para separar as células, chame o método de planilha unmerge_cells(). Digite o seguinte no shell interativo:

```
>>> import openpyxl
>>> wb = openpyxl.load_workbook('merged.xlsx')
```

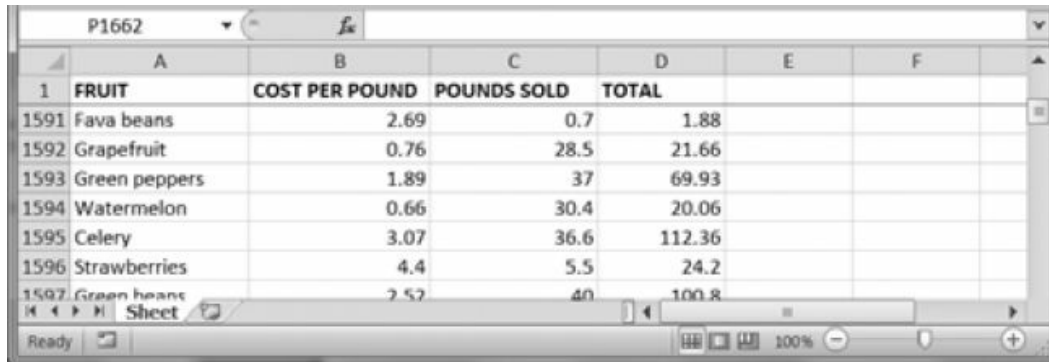
```
>>> sheet = wb.get_active_sheet()
>>> sheet.unmerge_cells('A1:D3')
>>> sheet.unmerge_cells('C5:D5')
>>> wb.save('merged.xlsx')
```

Se você salvar suas alterações e, em seguida, der uma olhada na planilha, verá que as células mescladas voltaram a ser células individuais.

Painéis congelados

Em planilhas grandes demais para serem exibidas de uma só vez, será conveniente "congelar" algumas das linhas da parte superior ou mais à esquerda na tela. Cabeçalhos de linhas ou colunas congeladas, por exemplo, permanecem sempre visíveis ao usuário mesmo que sejam feitas rolagens na planilha. Essas linhas e colunas congeladas são conhecidas como *painéis congelados* (freeze panes). No OpenPyXL, todo objeto Worksheet tem um atributo freeze_panes que pode ser definido com um objeto Cell ou com uma string das coordenadas da célula. Observe que todas as linhas acima e todas as colunas à esquerda dessa célula serão congeladas, porém a linha e a coluna da célula propriamente dita não serão congeladas.

Para descongelar todos os painéis, defina freeze_panes com None ou com 'A1'. A tabela 12.3 mostra quais linhas e colunas serão congeladas para alguns exemplos de configurações de freeze_panes.

Tabela 12.3 – Exemplos de painéis congelados

Configuração de freeze_panes	Linhas e colunas congeladas
sheet.freeze_panes = 'A2'	Linha 1
sheet.freeze_panes = 'B1'	Coluna A
sheet.freeze_panes = 'C1'	Colunas A e B
sheet.freeze_panes = 'C2'	Linha 1 e colunas A e B
sheet.freeze_panes = 'A1' ou sheet.freeze_panes = None	Não há painéis congelados

Certifique-se de ter a planilha de venda de produtos de *http://nostarch.com/automatestuff/*. Em seguida, digite o seguinte no shell interativo.

```
>>> import openpyxl
```

```
>>> wb = openpyxl.load_workbook('produceSales.xlsx')
>>> sheet = wb.get_active_sheet()
>>> sheet.freeze_panes = 'A2'
>>> wb.save('freezeExample.xlsx')
```

Se o atributo freeze_panes estiver definido com 'A2', a linha 1 sempre estará visível, independentemente de o usuário fazer rolagens na planilha. Isso pode ser visto na figura 12.8.

P1662

	A	B	C	D	E	F
1	FRUIT	COST PER POUND	POUNDS SOLD	TOTAL		
1591	Fava beans	2.69	0.7	1.88		
1592	Grapefruit	0.76	28.5	21.66		
1593	Green peppers	1.89	37	69.93		
1594	Watermelon	0.66	30.4	20.06		
1595	Celery	3.07	36.6	112.36		
1596	Strawberries	4.4	5.5	24.2		
1597	Green beans	2.52	40	100.8		

Sheet

Ready 100%

Figura 12.8 – Com freeze_panes *definido com* 'A2'*, a linha 1 sempre estará visível, mesmo que o usuário faça rolagens para baixo.*

Gráficos

O OpenPyXL suporta a criação de gráficos de barras, de linhas, de dispersão e de pizza usando dados das células de uma planilha. Para gerar um gráfico, você deve:

1. Criar um objeto Reference a partir de uma seleção retangular de células.
2. Criar um objeto Series passando-lhe o objeto Reference.
3. Criar um objeto Chart.
4. Adicionar o objeto Series ao objeto Chart.
5. Opcionalmente, definir as variáveis drawing.top, drawing.left, drawing.width e drawing.height do objeto Chart.
6. Adicionar o objeto Chart ao objeto Worksheet.

O objeto Reference exige um pouco de explicação. Os objetos Reference são criados por meio de chamadas à função openpyxl.charts.Reference(), passando-lhe três argumentos:

1. O objeto Worksheet contendo os dados de seu gráfico.
2. Uma tupla com dois inteiros que representa a célula superior esquerda da seleção retangular de células contendo os dados de seu gráfico: o primeiro inteiro da tupla é a linha e o segundo é a coluna. Observe que a primeira linha é 1, e não 0.
3. Uma tupla com dois inteiros que representa a célula inferior direita da seleção retangular de células contendo os dados de seu gráfico: o primeiro inteiro da tupla é a linha e o segundo é a coluna.

A figura 12.9 mostra alguns exemplos de argumentos de coordenadas.

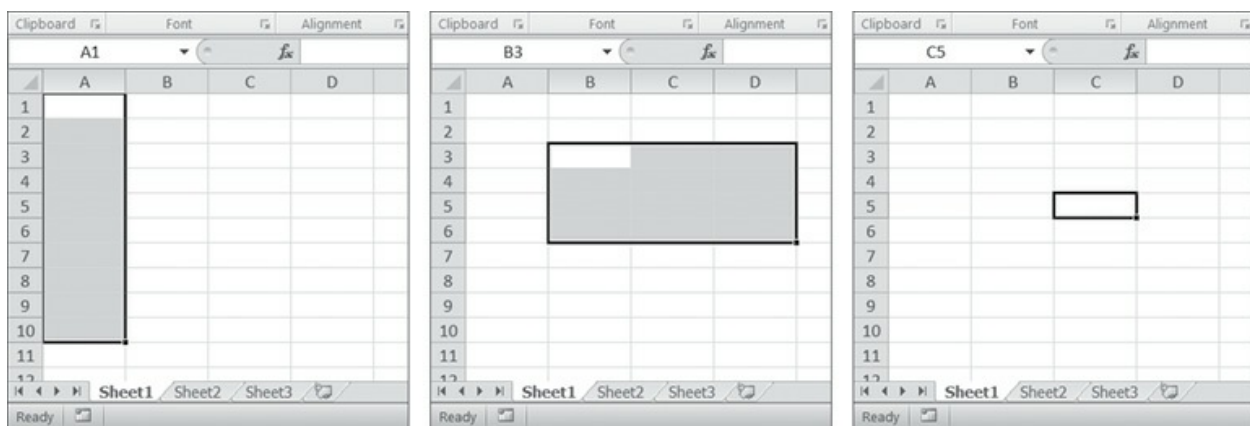



Figura 12.9 – Da esquerda para a direita: (1, 1), (10, 1)*;* (3, 2), (6, 4)*;* (5, 3), (5, 3).

Digite o exemplo a seguir no shell interativo para criar um gráfico de barras e adicioná-lo à planilha:

```
>>> import openpyxl
>>> wb = openpyxl.Workbook()
>>> sheet = wb.get_active_sheet()
>>> for i in range(1, 11):         # cria alguns dados na coluna A
        sheet['A' + str(i)] = i

>>> refObj = openpyxl.charts.Reference(sheet, (1, 1), (10, 1))

>>> seriesObj = openpyxl.charts.Series(refObj, title='First series')

>>> chartObj = openpyxl.charts.BarChart()
>>> chartObj.append(seriesObj)

>>> chartObj.drawing.top = 50       # define a posição
```

```
>>> chartObj.drawing.left = 100
>>> chartObj.drawing.width = 300    # define o tamanho
>>> chartObj.drawing.height = 200

>>> sheet.add_chart(chartObj)
>>> wb.save('sampleChart.xlsx')
```

Essas instruções geram uma planilha semelhante àquela mostrada na figura 12.10.

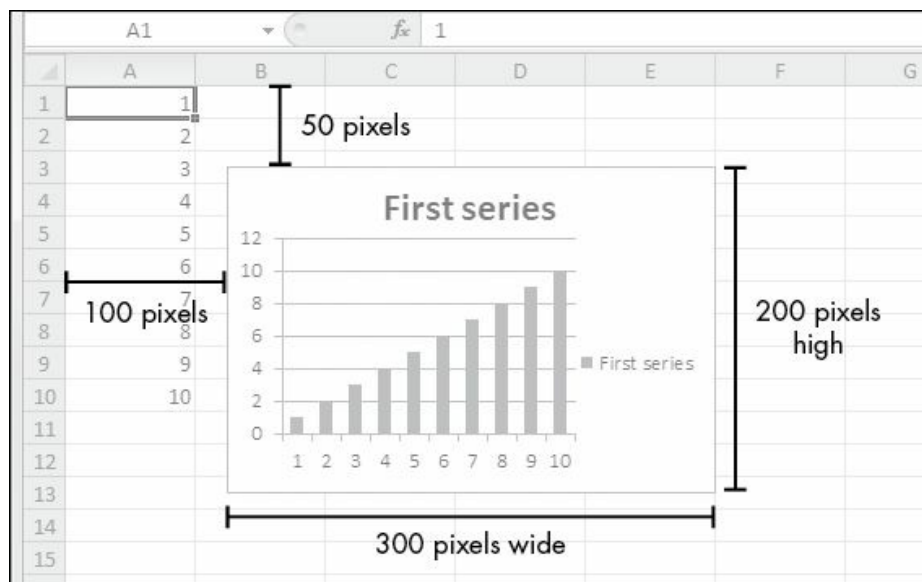



Figura 12.10 – Uma planilha com um gráfico adicionado.

Criamos um gráfico de barras ao chamar openpyxl.charts.BarChart(). Podemos criar também gráficos de linhas, de dispersão e de pizza chamando openpyxl.charts.LineChart(), openpyxl.charts.ScatterChart() e openpyxl.charts.PieChart().

Infelizmente, na versão atual do OpenPyXL (2.1.4), a função load_workbook() não carrega gráficos de arquivos Excel. Mesmo que o arquivo Excel tenha gráficos, o objeto Workbook carregado não os incluirá. Se você carregar um objeto Workbook e salvá-lo imediatamente no mesmo arquivo *.xlsx*, os gráficos serão efetivamente removidos desse arquivo.

Resumo

Geralmente, a parte difícil de processar informações não é o processamento em si, mas simplesmente obter os dados no formato correto para o seu

programa. Porém, depois que sua planilha estiver carregada em Python, será possível extrair e manipular seus dados muito mais rapidamente do que poderia ser feito manualmente.

Você também pode gerar planilhas como a saída de seus programas. Desse modo, se os seus colegas quiserem que seu arquivo-texto ou PDF contendo milhares de contatos de vendas seja transferido para um arquivo de planilha, você não terá a tarefa tediosa de copiar e colar tudo no Excel.

De posse do módulo openpyxl e de um pouco de conhecimento de programação, processar até mesmo as maiores planilhas será muito fácil.

Exercícios práticos

Para as perguntas a seguir, suponha que você tenha um objeto Workbook na variável wb, um objeto Worksheet em sheet, um objeto Cell em cell, um objeto Comment em comm e um objeto Image em img.

1. O que a função openpyxl.load_workbook() retorna?
2. O que o método de workbook get_sheet_names() retorna?
3. Como podemos obter o objeto Worksheet de uma planilha chamada 'Sheet1'?
4. Como podemos obter o objeto Worksheet da planilha ativa do workbook?
5. Como podemos obter o valor que está na célula C5?
6. Como podemos definir o valor da célula C5 para "Hello"?
7. Como podemos obter a linha e a coluna da célula na forma de inteiros?
8. O que os métodos de planilha get_highest_column() e get_highest_row() retornam e qual é o tipo de dado desses valores retornados?
9. Se for preciso obter o índice inteiro da coluna 'M', que função devemos chamar?
10. Se for preciso obter o nome em string da coluna 14, que função devemos chamar?
11. Como podemos obter uma tupla com todos os objetos Cell de A1 a F1?
12. Como podemos salvar o workbook em um arquivo chamado *example.xlsx*?
13. Como definimos uma fórmula em uma célula?

14. Se quisermos obter o resultado da fórmula de uma célula, e não a fórmula da célula, o que devemos fazer antes?
15. Como podemos definir a altura da linha 5 para 100?
16. Como podemos ocultar a coluna C?
17. Nomeie alguns recursos que o OpenPyXL 2.1.4 não carrega a partir de um arquivo de planilha.
18. O que é um painel congelado?
19. Quais são as cinco funções e os métodos que devemos chamar para criar um gráfico de barras?

Projetos práticos

Para exercitar, escreva programas que executem as tarefas a seguir.

Gerador de tabelas de multiplicação

Crie um programa *multiplicationTable.py* que receba um número *N* da linha de comando e crie uma tabela de multiplicação de *N*×*N* em uma planilha Excel. Por exemplo, quando o programa for executado desta maneira:

```
py multiplicationTable.py 6
```

ele deverá criar uma planilha como aquela mostrada na figura 12.11.

	A	B	C	D	E	F	G	H
1		**1**	**2**	**3**	**4**	**5**	**6**	
2	**1**	1	2	3	4	5	6	
3	**2**	2	4	6	8	10	12	
4	**3**	3	6	9	12	15	18	
5	**4**	4	8	12	16	20	24	
6	**5**	5	10	15	20	25	30	
7	**6**	6	12	18	24	30	36	
8								
9								

Figura 12.11 – Uma tabela de multiplicação gerada em uma planilha.

A linha 1 e a coluna A devem ser usadas como rótulos e devem estar em negrito.

Programa para inserção de linhas em branco

Crie um programa *blankRowInserter.py* que aceite dois inteiros e uma string

contendo um nome de arquivo como argumentos de linha de comando. Vamos chamar o primeiro inteiro de *N* e o segundo de *M*. Começando na linha *N*, o programa deve inserir *M* linhas em branco na planilha. Por exemplo, quando o programa for executado desta maneira:

```
python blankRowInserter.py 3 2 myProduce.xlsx
```

as planilhas "antes" e "depois" devem ter a aparência mostrada na figura 12.12.

Figura 12.12 – Antes (à esquerda) e depois (à direita) que as duas linhas em branco foram inseridas na linha 3.

Você pode escrever esse programa fazendo a leitura do conteúdo da planilha. Em seguida, ao gravar a nova planilha, utilize um loop for para copiar as *N* primeiras linhas. Para o restante das linhas, acrescente *M* ao número da linha na planilha de saída.

Programa para inverter células da planilha

Crie um programa para inverter a linha e a coluna das células da planilha. Por exemplo, o valor na linha 5, coluna 3 estará na linha 3, coluna 5 (e vice-versa). Isso deverá ser feito para todas as células da planilha. Por exemplo, as planilhas "antes" e "depois" devem ter a aparência mostrada na figura 12.13.

Você pode implementar esse programa usando loops for aninhados para ler os dados da planilha em uma estrutura de dados composta de lista de listas. Essa estrutura de dados pode conter sheetData[x][y] para a célula na coluna x e na linha y. Em seguida, ao escrever na nova planilha, utilize sheetData[y][x] para a célula na coluna x e na linha y.

A1 | fx | ITEM

	A	B	C	D	E	F	G	H	I	J
1	ITEM	SOLD								
2	Eggplant	334								
3	Cucumber	252								
4	Green cabl	238								
5	Eggplant	516								
6	Garlic	98								
7	Parsnips	16								
8	Asparagus	335								
9	Avocados	84								
10										

A1 | fx | ITEM

	A	B	C	D	E	F	G	H	I	J
1	ITEM	Eggplant	Cucumber	Green cabl	Eggplant	Garlic	Parsnips	Asparagus	Avocados	
2	SOLD	334	252	238	516	98	16	335	84	
3										
4										
5										
6										
7										
8										
9										
10										

Figura 12.13 – A planilha antes (na parte superior) e depois (na parte inferior) da inversão.

Arquivos-texto para planilha

Crie um programa que leia o conteúdo de diversos arquivos-texto (você mesmo pode criar esses arquivos) e insira seus conteúdos em uma planilha, com uma linha de texto em cada linha da planilha. As linhas do primeiro arquivo-texto estarão nas células da coluna A, as linhas do segundo arquivo-texto estarão nas células da coluna B e assim por diante.

Utilize o método readlines() do objeto File para retornar uma lista de strings do arquivo, com uma string por linha. Para o primeiro arquivo, insira a primeira linha na coluna 1, linha 1. A segunda linha deve ser escrita na coluna 1, linha 2, e assim sucessivamente. O próximo arquivo lido com readlines() será escrito na coluna 2, o próximo arquivo na coluna 3, e assim por diante.

Planilhas para arquivos-texto

Crie um programa que realize a tarefa do programa anterior na ordem

inversa: o programa deve abrir uma planilha e escrever as células da coluna A em um arquivo-texto, as células da coluna B em outro arquivo-texto e assim sucessivamente.

CAPÍTULO 13

TRABALHANDO COM DOCUMENTOS PDF E WORD

Os documentos PDF e Word são arquivos binários, o que os torna muito mais complexos do que os arquivos em formato texto simples. Além do texto, esses arquivos armazenam diversas informações sobre fonte, cor e layout. Se quiser que seus programas leiam ou escrevam em documentos PDF ou Word, será necessário fazer mais do que simplesmente passar seus nomes de arquivo para open().

Felizmente, existem módulos do Python que facilitam interagir com documentos PDF e Word. Este capítulo discutirá dois desses módulos: PyPDF2 e Python-Docx.

Documentos PDF

PDF que dizer *Portable Document Format*, e esse tipo de arquivo utiliza a extensão *.pdf*. Embora os PDFs suportem diversos recursos, este capítulo focará em duas tarefas que você fará com mais frequência com esses documentos: ler conteúdo textual dos PDFs e compor novos PDFs a partir de documentos existentes.

O módulo que usaremos para trabalhar com os PDFs é o PyPDF2. Para instalá-lo, execute pip install PyPDF2 na linha de comando. O nome desse módulo diferencia letras maiúsculas de minúsculas, portanto certifique-se de que o *y* seja minúsculo e tudo o mais esteja em letras maiúsculas. (Dê uma olhada no apêndice A para obter instruções detalhadas sobre a instalação de módulos de terceiros.) Se o módulo for instalado corretamente, a execução de import PyPDF2 no shell interativo não deverá exibir nenhuma mensagem de erro.

O FORMATO PDF PROBLEMÁTICO

Embora os arquivos PDF sejam ótimos para organizar textos de modo que seja fácil para as pessoas imprimirem e lerem, não é fácil para os softwares fazer parse e obter o texto em formato texto simples. Sendo assim, o PyPDF2 pode cometer erros ao extrair textos de um PDF e pode até mesmo ser incapaz de abrir alguns PDFs. Não há muito que se possa fazer a esse respeito, infelizmente. O PyPDF2 poderá simplesmente ser incapaz de trabalhar com alguns de seus arquivos PDF em particular. Apesar do que afirmei, ainda não encontrei nenhum arquivo PDF que não pudesse ser aberto com o PyPDF2.

Extraindo texto de PDFs

O PyPDF2 não tem uma maneira de extrair imagens, gráficos ou outros tipos de mídia de documentos PDF, porém pode extrair texto e retorná-lo na forma de uma string Python. Para começar a entender o funcionamento do PyPDF2, vamos usá-lo com o PDF de exemplo mostrado na figura 13.1.

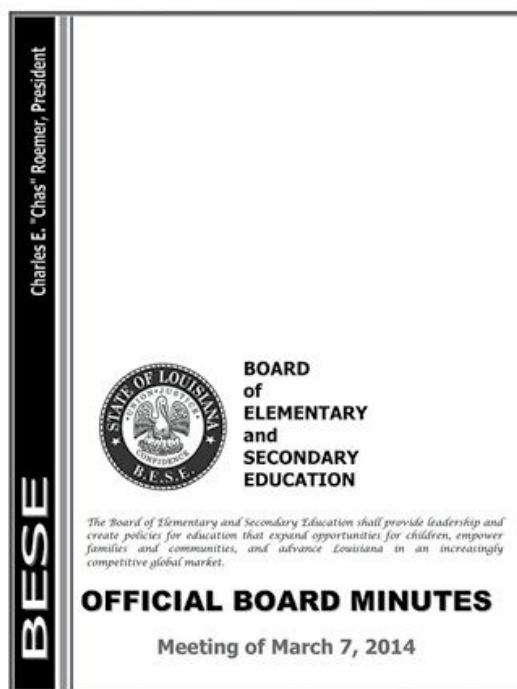



Figura 13.1 – A página de PDF da qual extrairemos o texto.

Faça o download desse PDF a partir de *http://nostarch.com/automatestuff/* e

digite o seguinte no shell interativo:

```
  >>> import PyPDF2
  >>> pdfFileObj = open('meetingminutes.pdf', 'rb')
  >>> pdfReader = PyPDF2.PdfFileReader(pdfFileObj)
u >>> pdfReader.numPages
  19
v >>> pageObj = pdfReader.getPage(0)
w >>> pageObj.extractText()
  'OOFFFFIICCIIAALL  BBOOAARRDD  MMIINNUUTTEESS   Meeting of March 7,
  2015       \n     The Board of Elementary and Secondary Education shall provide
  leadership and create policies for education that expand opportunities for children,
  empower families and communities, and advance Louisiana in an increasingly
  competitive global market. BOARD  of ELEMENTARY and  SECONDARY
  EDUCATION  '
```

Inicialmente, importe o módulo PyPDF2. Em seguida, abra *meetingminutes.pdf* em modo de leitura binária e armazene-o em pdfFileObj. Para obter um objeto PdfFileReader que represente esse PDF, chame PyPDF2.PdfFileReader() e passe-lhe o objeto pdfFileObj. Armazene esse objeto PdfFileReader em pdfReader.

O número total de páginas do documento é armazenado no atributo numPages de um objeto PdfFileReader u. O PDF de exemplo contém 19 páginas, porém vamos extrair o texto somente da primeira página.

Para extrair o texto de uma página, será necessário obter um objeto Page, que representa uma única página de um PDF, a partir de um objeto PdfFileReader. Um objeto Page pode ser obtido por meio da chamada ao método getPage() v em um objeto PdfFileReader, passando-lhe o número da página em que você estiver interessado – em nosso caso, é 0.

O PyPDF2 utiliza um *índice baseado em zero* para obter as páginas: a primeira página é 0, a segunda página é 1, e assim sucessivamente. A numeração sempre será dessa maneira, mesmo que as páginas estejam numeradas de forma diferente no documento. Por exemplo, suponha que o seu PDF seja um excerto de três páginas de um relatório mais longo e que suas páginas estejam numeradas como 42, 43 e 44. Para obter a primeira página desse documento, chame pdfReader.getPage(0), e não getPage(42) ou getPage(1).

Depois que tiver seu objeto Page, chame seu método extractText() para que uma string com o texto da página seja retornada w. A extração do texto não será perfeita: o texto *Charles E. "Chas" Roemer, President* do PDF não está presente na string retornada por extractText() e o espaçamento às vezes não coincide. Apesar disso, essa aproximação do conteúdo do texto PDF pode ser suficiente para o seu programa.

Descriptografando PDFs

Alguns documentos PDF têm um recurso de criptografia que evita que eles sejam lidos até que uma senha seja fornecida pela pessoa que estiver abrindo o documento. Digite o seguinte no shell interativo com o PDF baixado, que foi criptografado com a senha *rosebud*:

```
  >>> import PyPDF2
  >>> pdfReader = PyPDF2.PdfFileReader(open('encrypted.pdf', 'rb'))
u >>> pdfReader.isEncrypted
  True
  >>> pdfReader.getPage(0)
v Traceback (most recent call last):
    File "<pyshell#173>", line 1, in <module>
      pdfReader.getPage()
    --trecho removido--
    File "C:\Python34\lib\site-packages\PyPDF2\pdf.py", line 1173, in getObject
      raise utils.PdfReadError("file has not been decrypted")
  PyPDF2.utils.PdfReadError: file has not been decrypted
w >>> pdfReader.decrypt('rosebud')
  1
  >>> pageObj = pdfReader.getPage(0)
```

Todos os objetos PdfFileReader têm um atributo isEncrypted que será True se o PDF estiver criptografado e False se não estiver u. Qualquer tentativa de chamar uma função que leia o arquivo antes que ele seja descriptografado com a senha correta resultará em erro v.

Para ler um PDF criptografado, chame a função decrypt() e passe-lhe a senha na forma de uma string w. Após chamar decrypt() com a senha correta, você verá que chamar getPage() não causará mais um erro. Se a senha incorreta for especificada, a função decrypt() retornará 0 e getPage() continuará provocando

falhas. Observe que o método decrypt() descriptografa somente o objeto PdfFileReader, e não o arquivo PDF propriamente dito. Depois que seu programa for encerrado, o arquivo em seu disco rígido permanecerá criptografado. Seu programa deverá chamar decrypt() novamente na próxima vez que for executado.

Criando PDFs

A contrapartida do PyPDF2 ao objeto PdfFileReader é o objeto PdfFileWriter, que pode criar novos arquivos PDF. Porém o PyPDF2 não pode escrever qualquer texto em um PDF, como o Python pode fazer com arquivos em formato texto simples. As capacidades de escrita de PDF do PyPDF2 estão limitadas a copiar páginas de outros PDFs, fazer rotação de páginas, sobrepor páginas e criptografar arquivos.

O PyPDF2 não permite editar diretamente um PDF. Em vez disso, você deverá criar um novo PDF e então copiar o conteúdo de um documento existente. Os exemplos nesta seção seguirão a abordagem geral a seguir:

1. Abra um ou mais PDFs existentes (os PDFs originais) usando objetos PdfFileReader.
2. Crie um novo objeto PdfFileWriter.
3. Copie as páginas dos objetos PdfFileReader para o objeto PdfFileWriter.
4. Por fim, utilize o objeto PdfFileWriter para gravar o PDF de saída.

Criar um objeto PdfFileWriter fará somente um valor que representa um documento PDF ser criado em Python. Essa ação não criará o arquivo PDF propriamente dito. Para isso, chame o método write() de PdfFileWriter.

O método write() aceita um objeto File normal aberto em modo de *escrita binária*. Podemos obter um arquivo File desse tipo chamando a função open() do Python com dois argumentos: a string com o nome do arquivo PDF desejado e 'wb' para informar que o arquivo deve ser aberto em modo de escrita binária.

Se isso parecer um pouco confuso, não se preocupe – você verá como isso funciona nos exemplos de código a seguir.

Copiando páginas

Podemos usar PyPDF2 para copiar páginas de um documento PDF para outro. Isso permite combinar vários arquivos PDF, remover páginas indesejadas ou reordenar as páginas.

Faça download de *meetingminutes.pdf* e de *meetingminutes2.pdf* a partir de *http://nostarch.com/automatestuff/* e coloque os PDFs no diretório de trabalho atual. Digite o seguinte no shell interativo:

```
  >>> import PyPDF2
  >>> pdf1File = open('meetingminutes.pdf', 'rb')
  >>> pdf2File = open('meetingminutes2.pdf', 'rb')
u >>> pdf1Reader = PyPDF2.PdfFileReader(pdf1File)
v >>> pdf2Reader = PyPDF2.PdfFileReader(pdf2File)
w >>> pdfWriter = PyPDF2.PdfFileWriter()

  >>> for pageNum in range(pdf1Reader.numPages):
x         pageObj = pdf1Reader.getPage(pageNum)
y         pdfWriter.addPage(pageObj)

  >>> for pageNum in range(pdf2Reader.numPages):
x         pageObj = pdf2Reader.getPage(pageNum)
y         pdfWriter.addPage(pageObj)

z >>> pdfOutputFile = open('combinedminutes.pdf', 'wb')
  >>> pdfWriter.write(pdfOutputFile)
  >>> pdfOutputFile.close()
  >>> pdf1File.close()
  >>> pdf2File.close()
```

Abra ambos os arquivos PDF em modo de leitura binária e armazene os dois objetos File resultantes em pdf1File e em pdf2File. Chame PyPDF2.PdfFileReader() e passe pdf1File a fim de obter um objeto PdfFileReader para *meetingminutes.pdf* u. Chame essa função novamente e passe pdf2File a fim de obter um objeto PdfFileReader para *meetingminutes2.pdf* v. Em seguida, crie um novo objeto PdfFileWriter que representa um documento PDF vazio w.

A seguir, copie todas as páginas dos dois PDFs de origem e adicione-os ao objeto PdfFileWriter. Obtenha o objeto Page chamando getPage() em um objeto PdfFileReader x. Em seguida, passe esse objeto Page ao método addPage() de seu PdfFileWriter y. Esses passos são feitos inicialmente para pdf1Reader e

novamente para pdf2Reader. Quando acabar de copiar as páginas, gere um novo arquivo PDF chamado *combinedminutes.pdf* ao passar um objeto File para o método write() de PdfFileWriter z.

NOTA O PyPDF2 não é capaz de inserir páginas no meio de um objeto PdfFileWriter; o método addPage() adicionará páginas somente no final.

Agora você criou um novo arquivo PDF que combina as páginas de *meetingminutes.pdf* e de *meetingminutes2.pdf* em um único documento. Lembre-se de que o objeto File passado para PyPDF2.PdfFileReader() deve estar aberto em modo de leitura binária, o que é feito passando 'rb' como segundo argumento de open(). De modo semelhante, o objeto File passado para PyPDF2.PdfFileWriter() deve ser aberto em modo de escrita binária usando 'wb'.

Rotação de páginas

As páginas de um PDF também podem ser giradas em incrementos de 90 graus com os métodos rotateClockwise() e rotateCounterClockwise(). Passe um dos seguintes inteiros a esses métodos: 90, 180 ou 270. Digite o seguinte no shell interativo, com o arquivo *meetingminutes.pdf* no diretório de trabalho atual:

```
  >>> import PyPDF2
  >>> minutesFile = open('meetingminutes.pdf', 'rb')
  >>> pdfReader = PyPDF2.PdfFileReader(minutesFile)
u >>> page = pdfReader.getPage(0)
v >>> page.rotateClockwise(90)
  {'/Contents': [IndirectObject(961, 0), IndirectObject(962, 0),
  --trecho removido--
  }
  >>> pdfWriter = PyPDF2.PdfFileWriter()
  >>> pdfWriter.addPage(page)
w >>> resultPdfFile = open('rotatedPage.pdf', 'wb')
  >>> pdfWriter.write(resultPdfFile)
  >>> resultPdfFile.close()
  >>> minutesFile.close()
```

Nesse caso, utilizamos getPage(0) para selecionar a primeira página do PDF u e, em seguida, chamamos rotateClockwise(90) nessa página v. Criamos um novo PDF com a página girada e o salvamos como *rotatedPage.pdf* w.

O PDF resultante terá uma página girada em 90 graus no sentido horário,

conforme mostra a figura 13.2. Os valores de retorno de rotateClockwise() e de rotateCounterClockwise() contêm muitas informações que poderão ser ignoradas.

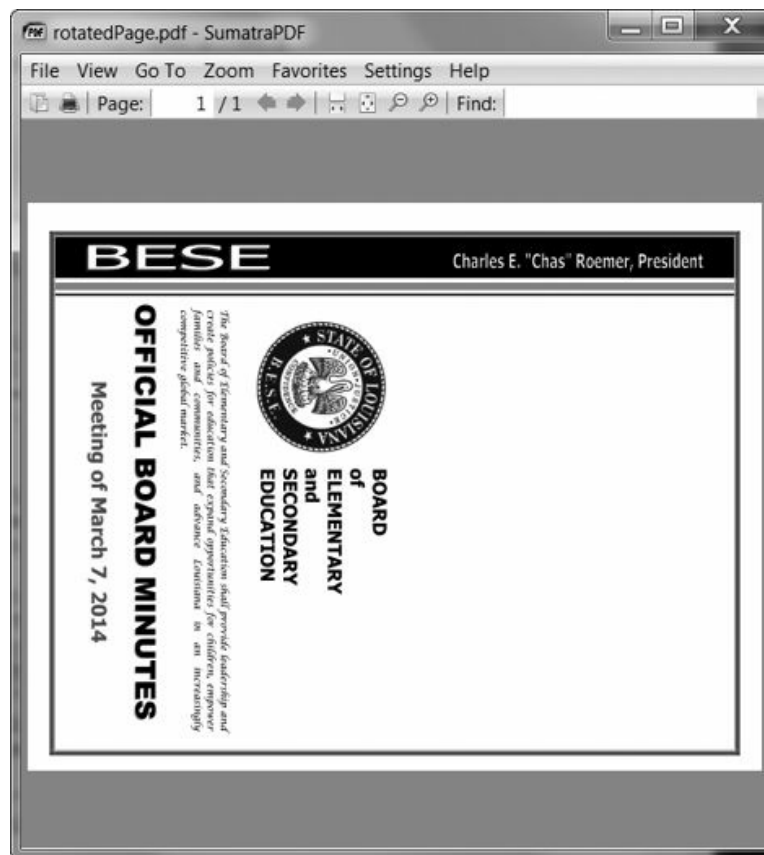



Figura 13.2 – O arquivo rotatedPage.pdf com a página girada em 90 graus no sentido horário.

Sobrepondo páginas

O PyPDF2 também pode sobrepor o conteúdo de uma página em outra, o que é útil para acrescentar um logo, um timestamp ou uma marca d'água em uma página. Em Python, é fácil adicionar marcas-d'água em diversos arquivos e somente nas páginas especificadas pelo seu programa.

Faça download de *watermark.pdf* a partir de *http://nostarch.com/automatestuff/* e coloque o PDF no diretório de trabalho atual, juntamente com *meetingminutes.pdf*. Depois, digite o seguinte no shell interativo.

```
  >>> import PyPDF2
  >>> minutesFile = open('meetingminutes.pdf', 'rb')
u >>> pdfReader = PyPDF2.PdfFileReader(minutesFile)
```

```
v >>> minutesFirstPage = pdfReader.getPage(0)
w >>> pdfWatermarkReader = PyPDF2.PdfFileReader(open('watermark.pdf', 'rb'))
x >>> minutesFirstPage.mergePage(pdfWatermarkReader.getPage(0))
y >>> pdfWriter = PyPDF2.PdfFileWriter()
z >>> pdfWriter.addPage(minutesFirstPage)

{ >>> for pageNum in range(1, pdfReader.numPages):
        pageObj = pdfReader.getPage(pageNum)
        pdfWriter.addPage(pageObj)

  >>> resultPdfFile = open('watermarkedCover.pdf', 'wb')
  >>> pdfWriter.write(resultPdfFile)
  >>> minutesFile.close()
  >>> resultPdfFile.close()
```

Nesse caso, criamos um objeto PdfFileReader a partir de *meetingminutes.pdf* u. Chamamos getPage(0) para obter um objeto Page da primeira página e armazenamos esse objeto em minutesFirstPage v. Em seguida, criamos um objeto PdfFileReader para *watermark.pdf* w e chamamos mergePage() em minutesFirstPage x. O argumento passado para mergePage() é um objeto Page da primeira página de *watermark.pdf*.

Agora que chamamos mergePage() em minutesFirstPage, minutesFirstPage representa a primeira página com a marca-d’água. Criamos um objeto PdfFileWriter y e adicionamos a primeira página com a marca-d’água z. Então percorremos o restante das páginas de *meetingminutes.pdf* em um loop e as adicionamos ao objeto PdfFileWriter {. Por fim, abrimos um novo PDF chamado *watermarkedCover.pdf* e gravamos o conteúdo de PdfFileWriter nesse novo arquivo PDF.

A figura 13.3 mostra o resultado. Nosso novo PDF, *watermarkedCover.pdf*, inclui todo o conteúdo de *meetingminutes.pdf* e a primeira página contém a marca-d’água.

Figura 13.3 – O PDF original (à esquerda), o PDF com a marca-d'água (no centro) e o PDF mesclado (à direita).

Criptografando PDFs

Um objeto PdfFileWriter também pode acrescentar criptografia a um documento PDF. Digite o seguinte no shell interativo:

```
>>> import PyPDF2
>>> pdfFile = open('meetingminutes.pdf', 'rb')
>>> pdfReader = PyPDF2.PdfFileReader(pdfFile)
>>> pdfWriter = PyPDF2.PdfFileWriter()
>>> for pageNum in range(pdfReader.numPages):
        pdfWriter.addPage(pdfReader.getPage(pageNum))

u >>> pdfWriter.encrypt('swordfish')
>>> resultPdf = open('encryptedminutes.pdf', 'wb')
>>> pdfWriter.write(resultPdf)
>>> resultPdf.close()
```

Antes de chamar o método write() para salvar dados em um arquivo, chame o método encrypt() e passe uma string de senha a ele u. Os PDFs podem ter uma *senha de usuário* (permite visualizar o PDF) e uma *senha de proprietário* (permite definir permissões para imprimir, comentar, extrair texto e outras funcionalidades). A senha de usuário e a senha de proprietário correspondem ao primeiro e ao segundo argumento de encrypt(), respectivamente. Se apenas um argumento de string for passado para encrypt(),

ele será usado para ambas as senhas.

Nesse exemplo, copiamos as páginas de *meetingminutes.pdf* para um objeto PdfFileWriter. Criptografamos PdfFileWriter com a senha *swordfish*, abrimos um novo PDF chamado *encryptedminutes.pdf* e gravamos o conteúdo de PdfFileWriter no novo PDF. Antes que alguém possa ver *encryptedminutes.pdf*, essa senha deverá ser fornecida. Você pode apagar o arquivo *meetingminutes.pdf* original, que não está criptografado, após garantir que sua cópia foi corretamente criptografada.

Projeto: Combinando páginas selecionadas de vários PDFs

Suponha que você tenha a tarefa maçante de combinar diversos documentos PDF em um único arquivo PDF. Cada um deles tem uma folha de rosto na primeira página, porém você não quer que essa folha seja repetida no resultado final. Apesar de haver vários programas gratuitos para combinar PDFs, muitos deles simplesmente combinam os arquivos inteiros. Vamos criar um programa Python que personalize quais páginas queremos ter no PDF combinado.

De modo geral, eis o que o programa deverá fazer:

- Encontrar todos os arquivos PDF no diretório de trabalho atual.
- Ordenar os nomes dos arquivos para que os PDFs sejam acrescentados na sequência.
- Gravar as páginas de cada PDF, exceto a primeira, no arquivo de saída.

Em termos de implementação, seu código deverá fazer o seguinte:

- Chamar os.listdir() para encontrar todos os arquivos no diretório de trabalho atual e excluir qualquer arquivo que não seja PDF.
- Chamar o método de lista sort() do Python para organizar os nomes dos arquivos em ordem alfabética.
- Criar um objeto PdfFileWriter para o PDF de saída.
- Percorrer todos os arquivos PDF em um loop, criando um objeto PdfFileReader para cada um.
- Percorrer as páginas (exceto a primeira) de cada arquivo PDF em um loop.

• Acrescentar as páginas ao PDF de saída.
• Gravar o PDF de saída em um arquivo chamado *allminutes.pdf*.

Para esse projeto, abra uma nova janela no editor de arquivo e salve o programa como *combinePdfs.py*.

Passo 1: Encontrar todos os arquivos PDF

Inicialmente, seu programa deve obter uma lista de todos os arquivos com extensão *.pdf* no diretório de trabalho atual e ordená-la. Faça o seu código ter o seguinte aspecto:

```
   #! python3
   # combinePdfs.py – Combina todos os PDFs do diretório de trabalho atual em
   # um único PDF.

u import PyPDF2, os

   # Obtém os nomes de todos os arquivos PDF.
   pdfFiles = []
   for filename in os.listdir('.'):
      if filename.endswith('.pdf'):
v        pdfFiles.append(filename)
w pdfFiles.sort(key=str.lower)

x pdfWriter = PyPDF2.PdfFileWriter()

   # TODO: Percorre todos os arquivos PDF em um loop.

   # TODO: Percorre todas as páginas (exceto a primeira) e as adiciona no PDF de saída.

   # TODO: Salva o PDF resultante em um arquivo.
```

Após a linha shebang e o comentário descritivo sobre o que o programa faz, esse código importa os módulos os e PyPDF2 u. A chamada a os.listdir('.') retorna uma lista de todos os arquivos no diretório de trabalho atual. O código percorre essa lista em um loop e adiciona somente os arquivos cuja extensão seja *.pdf* em pdfFiles v. Depois disso, essa lista é organizada em ordem alfabética com o argumento nomeado key=str.lower de sort() w.

Um objeto PdfFileWriter é criado para armazenar as páginas PDF

combinadas x. Por fim, alguns comentários descrevem o restante do programa.

Passo 2: Abrir cada PDF

Agora o programa deve ler cada arquivo PDF que estiver em pdfFiles. Acrescente o seguinte em seu programa:

```
#! python3
# combinePdfs.py – Combina todos os PDFs do diretório de trabalho atual em
# um único PDF.

import PyPDF2, os

# Obtém os nomes de todos os arquivos PDF.
pdfFiles = []
--trecho removido--

# Percorre todos os arquivos PDF em um loop.
for filename in pdfFiles:
   pdfFileObj = open(filename, 'rb')
   pdfReader = PyPDF2.PdfFileReader(pdfFileObj)
   # TODO: Percorre todas as páginas (exceto a primeira) e as adiciona no PDF de saída.

# TODO: Salva o PDF resultante em um arquivo.
```

Para cada PDF, o loop abre um arquivo em modo de leitura binária ao chamar open() com 'rb' como segundo argumento. A chamada a open() retorna um objeto File que é passado para PyPDF2.PdfFileReader() a fim de criar um objeto PdfFileReader para esse arquivo PDF.

Passo 3: Adicionar cada página

Para cada PDF, você deverá percorrer todas as páginas, exceto a primeira, em um loop. Acrescente o código a seguir em seu programa:

```
#! python3
# combinePdfs.py – Combina todos os PDFs do diretório de trabalho atual em
# um único PDF.

import PyPDF2, os
```

```
--trecho removido--

# Percorre todos os arquivos PDF em um loop.
for filename in pdfFiles:
--trecho removido--
   # Percorre todas as páginas (exceto a primeira) e as adiciona no PDF de saída.
u  for pageNum in range(1, pdfReader.numPages):
       pageObj = pdfReader.getPage(pageNum)
       pdfWriter.addPage(pageObj)

# TODO: Salva o PDF resultante em um arquivo.
```

O código no loop for copia cada objeto Page individualmente para o objeto PdfFileWriter. Lembre-se de que a primeira página deve ser ignorada. Como PyPDF2 considera a página 0 como sendo a primeira, seu loop deve iniciar em 1 u e, em seguida, esse valor deve ser incrementado até atingir o inteiro em pdfReader.numPages, porém sem incluí-lo.

Passo 4: Salvar o resultado

Após esses loops for aninhados terem sido concluídos, a variável pdfWriter conterá um objeto PdfFileWriter com as páginas de todos os PDFs combinados. O último passo consiste em gravar esse conteúdo em um arquivo no disco rígido. Acrescente o código a seguir em seu programa:

```
#! python3
# combinePdfs.py – Combina todos os PDFs do diretório de trabalho atual em
# um único PDF.

import PyPDF2, os

--trecho removido--

# Percorre todos os arquivos PDF em um loop.
for filename in pdfFiles:
--trecho removido--
   # Percorre todas as páginas (exceto a primeira) e as adiciona no PDF de saída.
     for pageNum in range(1, pdfReader.numPages):
   --trecho removido--
```

```
# Salva o PDF resultante em um arquivo.
pdfOutput = open('allminutes.pdf', 'wb')
pdfWriter.write(pdfOutput)
pdfOutput.close()
```

Passar 'wb' para open() faz o arquivo PDF *allminutes.pdf* de saída ser aberto em modo de escrita binária. Então passar o objeto File resultante ao método write() fará o arquivo PDF propriamente dito ser criado. Uma chamada ao método close() encerra o programa.

Ideias para programas semelhantes

Ser capaz de criar PDFs a partir de páginas de outros PDFs permitirá criar programas que possam:

- Remover páginas específicas de PDFs.
- Reorganizar as páginas em um PDF.
- Criar um PDF apenas com as páginas que tenham um texto específico identificado por extractText().

Documentos Word

O Python pode criar e modificar documentos Word, que têm extensão de arquivo *.docx*, com o módulo python-docx. Esse módulo pode ser instalado por meio da execução de pip install python-docx. (O apêndice A contém os detalhes completos sobre a instalação de módulos de terceiros.)

NOTA Ao usar pip para instalar inicialmente o Python-Docx, certifique-se de que instalará python-docx, e não docx. O nome de instalação docx é usado para um módulo diferente, que não será discutido neste livro. Entretanto, ao importar o módulo python-docx, será necessário executar import docx, e não import python-docx.

Se você não tiver o Word, o LibreOffice Writer e o OpenOffice Writer são alternativas de aplicativos gratuitos para Windows, OS X e Linux e podem ser usados para abrir arquivos *.docx*. O download desses aplicativos pode ser feito a partir de *https://www.libreoffice.org* e de *http://openoffice.org*, respectivamente. A documentação completa do Python-Docx está disponível

em *https://python-docx.readthedocs.org/*. Embora haja uma versão de Word para OS X, este capítulo focará no Word para Windows.

Comparados aos arquivos em formato texto simples, os arquivos *.docx* contêm muitas estruturas. Essas estruturas são representadas por três tipos de dados diferentes no Python-Docx. Em âmbito geral, um objeto Document representa o documento completo. O objeto Document contém uma lista de objetos Paragraph para os parágrafos do documento. (Um novo parágrafo começa sempre que o usuário tecla enter ou return enquanto está digitando um documento Word.) Cada um desses objetos Paragraph contém uma lista de um ou mais objetos Run. O parágrafo com uma única sentença na figura 13.4 contém quatro runs.

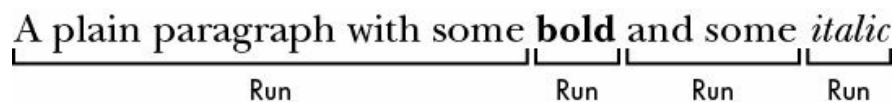



Figura 13.4 – Os objetos Run *identificados em um objeto* Paragraph.

O texto em um documento Word é mais que apenas uma string. Esse texto tem fonte, tamanho, cor e outras informações de estilização associadas a ele. Um *estilo* (style) em Word é uma coleção desses atributos. Um objeto Run é uma porção contígua de texto que apresenta o mesmo estilo. Um novo objeto Run é necessário sempre que o estilo do texto mudar.

Lendo documentos Word

Vamos testar o módulo python-docx. Faça download de *demo.docx* a partir de *http://nostarch.com/automatestuff/* e salve o documento no diretório de trabalho. Depois, digite o seguinte no shell interativo.

```
  >>> import docx
u >>> doc = docx.Document('demo.docx')
v >>> len(doc.paragraphs)
  7
w >>> doc.paragraphs[0].text
  'Document Title'
x >>> doc.paragraphs[1].text
  'A plain paragraph with some bold and some italic'
y >>> len(doc.paragraphs[1].runs)
  4
```

```
z >>> doc.paragraphs[1].runs[0].text
  'A plain paragraph with some '
{ >>> doc.paragraphs[1].runs[1].text
  'bold'
| >>> doc.paragraphs[1].runs[2].text
  ' and some '
} >>> doc.paragraphs[1].runs[3].text
  'italic'
```

Em u abrimos um arquivo *.docx* em Python, chamamos docx.Document() e passamos o nome do arquivo *demo.docx*. Essa chamada retorna um objeto Document com um atributo paragraphs, que é uma lista de objetos Paragraph. Quando chamamos len() em doc.paragraphs, essa função retorna 7, que nos informa que há sete objetos Paragraph nesse documento v. Cada um desses objetos Paragraph tem um atributo text que contém uma string do texto nesse parágrafo (sem as informações de estilo). Nesse caso, o primeiro atributo text contém 'DocumentTitle' w e o segundo contém 'A plain paragraph with some bold and some italic' x.

Cada objeto Paragraph também tem um atributo runs que é uma lista de objetos Run. Os objetos Run também têm um atributo text que contém somente o texto desse run em particular. Vamos dar uma olhada no atributo text do segundo objeto Paragraph, ou seja, 'A plain paragraph with some bold and some italic'. Chamar len() nesse objeto Paragraph nos informa que há quatro objetos Run y. O primeiro objeto run contém 'A plain paragraph with some ' z. Em seguida, o texto muda para um estilo em negrito, portanto 'bold' inicia um novo objeto Run {. O texto retorna para um estilo sem negrito depois disso, resultando em um terceiro objeto Run com ' and some ' |. Por fim, o quarto e último objeto Run contém 'italic' em um estilo itálico }.

Com o Python-Docx, seus programas Python poderão ler o texto de um arquivo *.docx* e usá-lo como se fosse um valor de string qualquer.

Obtendo o texto completo de um arquivo .docx

Se você estiver interessado somente no texto e não nas informações de estilização do documento Word, a função getText() poderá ser usada. Ela aceita o nome de um arquivo *.docx* e retorna um único valor de string

contendo seu texto. Abra uma nova janela no editor de arquivo e insira o código a seguir, salvando-o como *readDocx.py*:

```
#! python3

import docx

def getText(filename):
    doc = docx.Document(filename)
    fullText = []
    for para in doc.paragraphs:
        fullText.append(para.text)
    return '\n'.join(fullText)
```

A função getText() abre o documento Word, percorre todos os objetos Paragraph da lista paragraphs em um loop e adiciona seus textos à lista em fullText. Após o loop, as strings em fullText são unidas com caracteres de quebra de linha.

O programa *readDocx.py* pode ser importado como qualquer outro módulo. Se você precisar somente do texto de um documento Word, poderá digitar o seguinte:

```
>>> import readDocx
>>> print(readDocx.getText('demo.docx'))
Document Title
A plain paragraph with some bold and some italic
Heading, level 1
Intense quote
first item in unordered list
first item in ordered list
```

Podemos também ajustar getText() para modificar a string antes de retorná-la. Por exemplo, para indentar cada parágrafo, substitua a chamada a append() em *readDocx.py* por:

```
fullText.append('  ' + para.text)
```

Para acrescentar um espaço duplo entre os parágrafos, altere a chamada a join() por:

```
return '\n\n'.join(fullText)
```

Como podemos ver, somente algumas linhas de código são necessárias para criar funções que leiam um arquivo *.docx* e retornem uma string com seu conteúdo de acordo com a sua preferência.

Estilizando parágrafos e objetos Run

No Word para Windows, podemos ver os estilos ao pressionar CTRL-ALT-SHIFT-S para exibir o painel Styles (Estilos), que tem um aspecto semelhante ao mostrado na figura 13.5. No OS X, podemos visualizar o painel Styles clicando no item de menu View4Styles (Visualizar4Estilos).

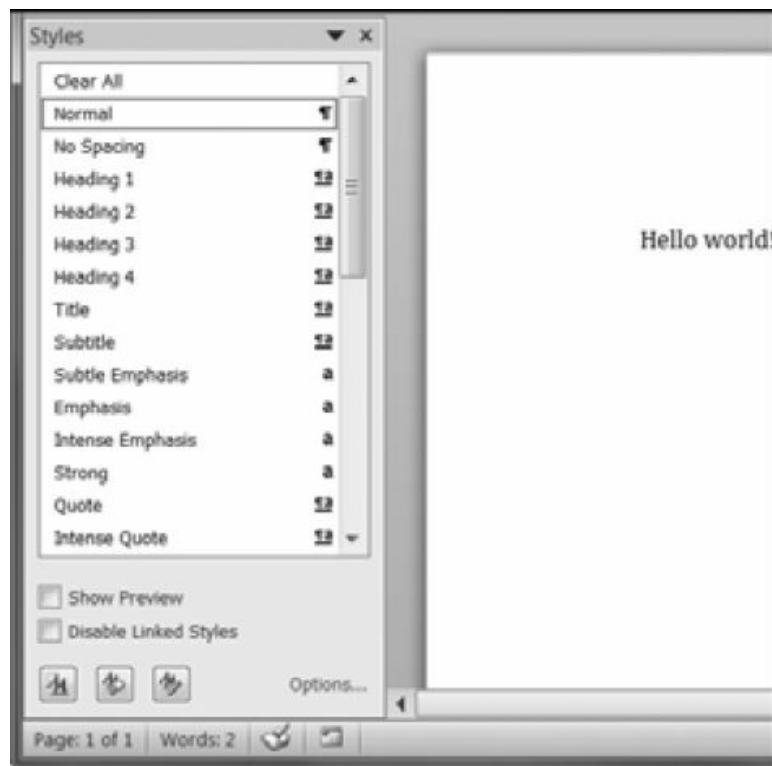



Figura 13.5 – O painel Styles (Estilos) é exibido com CTRL-ALT-SHIFT-*S no Windows.*

O Word e outros processadores de texto usam estilos para que a apresentação visual de tipos semelhantes de texto permaneça consistente e fácil de alterar. Por exemplo, talvez você queira definir o corpo dos parágrafos com texto de 11 pontos em Times New Roman, justificado à esquerda, sem alinhamento à direita. Podemos criar um estilo com essas configurações e atribuí-lo a todos os parágrafos. Então, se você quiser alterar a apresentação de todos os parágrafos do documento posteriormente, poderá simplesmente alterar o estilo e todos os parágrafos serão automaticamente

atualizados.

Em documentos Word, há três tipos de estilo: *estilos de parágrafo* (paragraph styles), que podem ser aplicados em objetos Paragraph, *estilos de caractere* (character styles), que podem ser aplicados a objetos Run, e *estilos vinculados* (linked styles), que podem ser aplicados a ambos os tipos de objeto. Podemos atribuir estilos tanto aos objetos Paragraph quanto aos objetos Run ao definir seus atributos style com uma string. Essa string deve conter o nome de um estilo. Se style estiver definido com None, não haverá nenhum estilo associado ao objeto Paragraph ou Run.

Os valores de string para os estilos default do Word são:

'Normal'	'Heading5'	'ListBullet'	
'BodyText'	'Heading6'	'ListBullet2'	'ListParagraph'
'BodyText2'	'Heading7'	'ListBullet3'	'MacroText'
'BodyText3'	'Heading8'	'ListContinue'	'NoSpacing'
'Caption'	'Heading9'	'ListContinue2'	'Quote'
'Heading1'	'IntenseQuote'	'ListContinue3'	'Subtitle'
'Heading2'	'List'	'ListNumber'	'TOCHeading'
'Heading3'	'List2'	'ListNumber2'	'Title'
'Heading4'	'List3'	'ListNumber3'	

Ao definir o atributo style, não utilize espaços no nome do estilo. Por exemplo, embora o nome do estilo possa ser Subtle Emphasis, defina o atributo style com o valor de string 'SubtleEmphasis' em vez de 'Subtle Emphasis'. Incluir espaços fará o Word ler incorretamente o nome do estilo, e ele não será aplicado.

Ao usar um estilo vinculado em um objeto Run, será necessário acrescentar 'Char' no final de seu nome. Por exemplo, para definir o estilo vinculado Quote em um objeto Paragraph, utilize paragraphObj.style = 'Quote', mas, para um objeto Run, utilize runObj.style = 'QuoteChar'.

Na versão atual do Python-Docx (0.7.4), os únicos estilos que podem ser usados são os estilos default do Word e os estilos do *.docx* aberto. Novos estilos não poderão ser criados – embora isso possa mudar nas futuras versões do Python-Docx.

Criando documentos Word com estilos que não sejam default

Se quiser criar documentos Word que usem estilos que não sejam os estilos default, será necessário abrir o Word com um documento em branco e criar os estilos por conta própria clicando no botão New Style (Novo estilo) na parte inferior do painel Styles (Estilos) – veja a figura 13.6, que mostra isso no Windows.

Isso fará o diálogo Create New Style from Formatting (Criar Novo Estilo a Partir da Formatação) ser apresentado para que você possa criar um novo estilo. Em seguida, retorne ao shell interativo e abra esse documento Word em branco com docx.Document(); use-o como base para seu documento Word. O nome dado a esse estilo não estará disponível para ser usado com o Python-Docx.

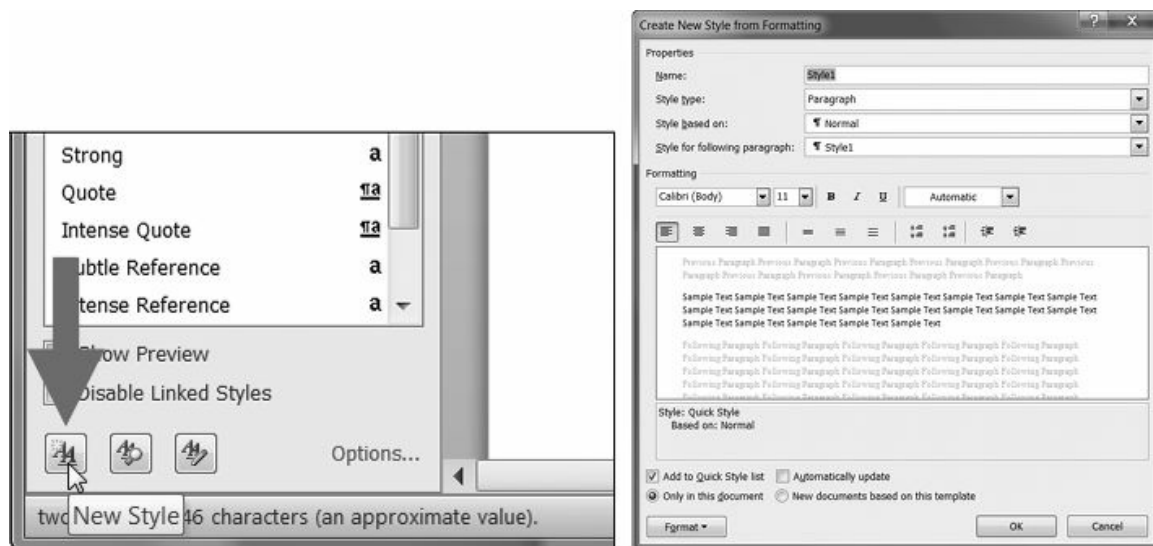



Figura 13.6 – O botão New Style (Novo estilo) à esquerda e o diálogo Create New Style from Formatting (Criar Novo Estilo a Partir da Formatação) à direita.

Atributos de Run

Os runs podem ser estilizados por meio de atributos de text. Cada atributo pode ser definido com um de três valores: True (o atributo está sempre habilitado, independentemente de outros estilos aplicados ao run), False (o atributo está sempre desabilitado) ou None (usa qualquer estilo definido no run como default).

A tabela 13.1 lista os atributos de text que podem ser definidos em objetos Run.

Tabela 13.1 – Atributos de text *do objeto* Run

Atributo	Descrição
bold	O texto aparece em negrito.
italic	O texto aparece em itálico.
underline	O texto é sublinhado.
strike	O texto aparece com uma linha no meio (tachado).
double_strike	O texto aparece com duas linhas no meio (tachado duplo).
all_caps	O texto aparece em letras maiúsculas.
small_caps	O texto aparece em letras maiúsculas e as letras minúsculas têm dois pontos a menos.
shadow	O texto aparece sombreado.
outline	O texto aparece contornado em vez de ser sólido.
rtl	O texto é escrito da direita para a esquerda.
imprint	O texto aparece em profundidade na página.
emboss	O texto aparece em relevo na página.

Por exemplo, para alterar os estilos de *demo.docx*, digite o seguinte no shell interativo:

```
>>> doc = docx.Document('demo.docx')
>>> doc.paragraphs[0].text
'Document Title'
>>> doc.paragraphs[0].style
'Title'
>>> doc.paragraphs[0].style = 'Normal'
>>> doc.paragraphs[1].text
'A plain paragraph with some bold and some italic'
>>> (doc.paragraphs[1].runs[0].text, doc.paragraphs[1].runs[1].text,
doc.paragraphs[1].runs[2].text, doc.paragraphs[1].runs[3].text)
('A plain paragraph with some ', 'bold', ' and some ', 'italic')
>>> doc.paragraphs[1].runs[0].style = 'QuoteChar'
>>> doc.paragraphs[1].runs[1].underline = True
>>> doc.paragraphs[1].runs[3].underline = True
>>> doc.save('restyled.docx')
```

Nesse caso, usamos os atributos text e style para ver facilmente o que os parágrafos de nosso documento contêm. Podemos ver que é fácil dividir um parágrafo em runs e acessar cada run individualmente. Desse modo,

obtivemos o primeiro, o segundo e o quarto runs do segundo parágrafo, estilizamos cada run e salvamos o resultado em um novo documento.

As palavras *Document Title* no início de *restyled.docx* terão o estilo Normal no lugar do estilo Title, o objeto Run do texto *A plain paragraph with some* terá o estilo QuoteChar e os dois objetos Run para as palavras *bold* e *italic* terão seus atributos underline definidos com True. A figura 13.7 mostra a aparência dos estilos dos parágrafos e dos runs em *restyled.docx*.

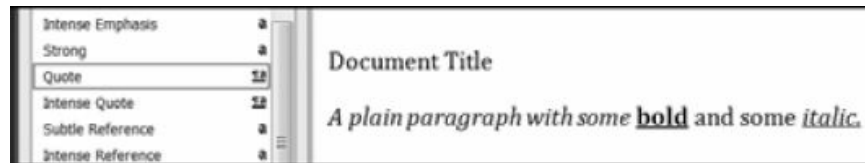



Figura 13.7 – O arquivo restyled.docx.

Você poderá encontrar uma documentação mais completa sobre o uso de estilos no Python-Docx em *https://python-docx.readthedocs.org/en/latest/user/styles.html*.

Escrevendo em documentos Word

Digite o seguinte no shell interativo:

```
>>> import docx
>>> doc = docx.Document()
>>> doc.add_paragraph('Hello world!')
<docx.text.Paragraph object at 0x0000000003B56F60>
>>> doc.save('helloworld.docx')
```

Para criar seu próprio arquivo *.docx*, chame docx.Document() para retornar um novo objeto Document do Word em branco. O método de documento add_paragraph() acrescenta um novo parágrafo de texto no documento e retorna uma referência ao objeto Paragraph adicionado. Quando acabar de adicionar textos, passe uma string com um nome de arquivo ao método de documento save() para salvar o objeto Document em um arquivo.

Isso fará um arquivo chamado *helloworld.docx* ser criado no diretório de trabalho atual; quando aberto, esse documento terá a aparência mostrada na figura 13.8.

Figura 13.8 – O documento Word criado com add_paragraph('Hello world!').

Podemos adicionar parágrafos ao chamar o método add_paragraph() novamente com o texto do novo parágrafo. Para adicionar texto no final de um parágrafo existente, podemos chamar o método add_run() do parágrafo e passar-lhe uma string. Digite o seguinte no shell interativo:

```
>>> import docx
>>> doc = docx.Document()
>>> doc.add_paragraph('Hello world!')
<docx.text.Paragraph object at 0x000000000366AD30>
>>> paraObj1 = doc.add_paragraph('This is a second paragraph.')
>>> paraObj2 = doc.add_paragraph('This is a yet another paragraph.')
>>> paraObj1.add_run(' This text is being added to the second paragraph.')
<docx.text.Run object at 0x0000000003A2C860>
>>> doc.save('multipleParagraphs.docx')
```

O documento resultante terá a aparência mostrada na figura 13.9. Observe que o texto *This text is being added to the second paragraph.* foi adicionado ao objeto Paragraph em paraObj1, que foi o segundo parágrafo adicionado a doc. As funções add_paragraph() e add_run() retornam objetos Paragraph e Run, respectivamente, para evitar que você tenha o trabalho de extraí-los em um passo separado.

Tenha em mente que, na versão 0.5.3 do Python-Docx, novos objetos Paragraph podem ser adicionados somente no final do documento e novos objetos Run podem ser adicionados somente no final de um objeto Paragraph.

O método save() pode ser chamado novamente para salvar as alteração adicionais feitas por você.

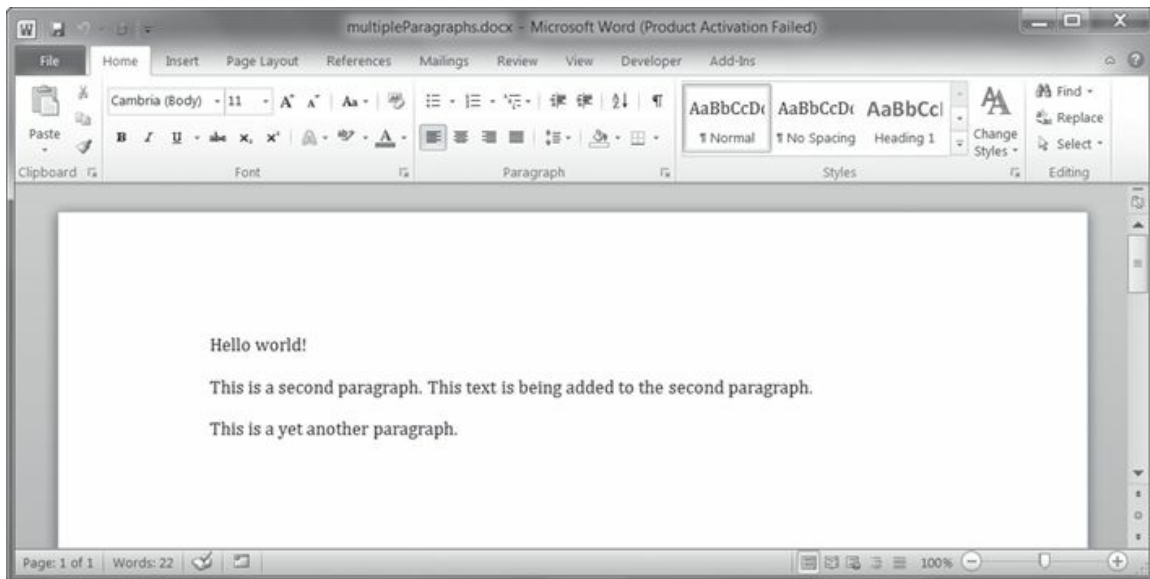



Figura 13.9 – O documento com vários objetos Paragraph *e* Run *adicionados.*

Tanto add_paragraph() quanto add_run() aceitam um segundo argumento opcional composto de uma string com o estilo do objeto Paragraph ou Run. Por exemplo:

```
>>> doc.add_paragraph('Hello world!', 'Title')
```

Essa linha acrescenta um parágrafo com o texto *Hello world!* no estilo Title.

Adicionando títulos

Chamar add_heading() adiciona um parágrafo com um dos estilos para títulos. Digite o seguinte no shell interativo:

```
>>> doc = docx.Document()
>>> doc.add_heading('Header 0', 0)
<docx.text.Paragraph object at 0x00000000036CB3C8>
>>> doc.add_heading('Header 1', 1)
<docx.text.Paragraph object at 0x00000000036CB630>
>>> doc.add_heading('Header 2', 2)
<docx.text.Paragraph object at 0x00000000036CB828>
>>> doc.add_heading('Header 3', 3)
```

```
<docx.text.Paragraph object at 0x00000000036CB2E8>
>>> doc.add_heading('Header 4', 4)
<docx.text.Paragraph object at 0x00000000036CB3C8>
>>> doc.save('headings.docx')
```

Os argumentos de add_heading() são uma string com o texto do título e um inteiro de 0 a 4. O inteiro 0 deixa o título com estilo Title, que é usado para o nível mais alto do documento. Os inteiros de 1 a 4 servem para vários níveis de títulos, em que 1 é o título principal e 4 é o subtítulo de nível mais baixo. A função add_heading() retorna um objeto Paragraph para evitar que você precise extraí-lo do objeto Document em um passo separado.

O arquivo *headings.docx* resultante terá a aparência mostrada na figura 13.10.

Figura 13.10 – O documento headings.docx com títulos de 0 a 4.

Adicionando quebras de linha e de página

Para adicionar uma quebra de linha (em vez de iniciar todo um novo parágrafo), podemos chamar o método add_break() no objeto Run em que você quer que a quebra apareça. Se quiser adicionar uma quebra de página em seu lugar, passe o valor docx.text.WD_BREAK.PAGE como o único argumento de add_break(), como está sendo mostrado no meio do exemplo a seguir:

```
  >>> doc = docx.Document()
  >>> doc.add_paragraph('This is on the first page!')
  <docx.text.Paragraph object at 0x0000000003785518>
u >>> doc.paragraphs[0].runs[0].add_break(docx.text.WD_BREAK.PAGE)
  >>> doc.add_paragraph('This is on the second page!')
  <docx.text.Paragraph object at 0x00000000037855F8>
  >>> doc.save('twoPage.docx')
```

Isso cria um documento Word de duas páginas, com *This is on the first page!* na primeira página e *This is on the second page!* na segunda. Apesar de ainda haver bastante espaço na primeira página após o texto *This is on the first page!*, forçamos o próximo parágrafo a começar em uma nova página ao inserir uma quebra de página após o primeiro run do primeiro parágrafo u.

Adicionando imagens

Os objetos Document têm um método add_picture() que permite adicionar uma imagem no final do documento. Suponha que você tenha um arquivo *zophie.png* no diretório de trabalho atual. Você pode adicionar *zophie.png* no final de seu documento com largura de uma polegada e altura de quatro centímetros (o Word pode usar tanto unidades métricas quanto imperiais) se digitar o seguinte:

```
>>> doc.add_picture('zophie.png', width=docx.shared.Inches(1),
height=docx.shared.Cm(4))
<docx.shape.InlineShape object at 0x00000000036C7D30>
```

O primeiro argumento é uma string com o nome do arquivo de imagem. Os argumentos nomeados opcionais width e height definirão a largura e a altura da imagem no documento. Se não forem especificados, a largura e a altura serão definidas com o tamanho normal da imagem como default.

Provavelmente, você preferirá especificar a altura e a largura de uma imagem com unidades com as quais tiver familiaridade, por exemplo, polegadas e centímetros; desse modo, as funções docx.shared.Inches() e docx.shared.Cm() poderão ser usadas quando os argumentos nomeados width e height forem especificados.

Resumo

Informações textuais não são usadas apenas em arquivos com formato texto simples; de fato, é bem provável que você vá lidar com documentos PDF e Word com muito mais frequência. O módulo PyPDF2 pode ser usado para ler e escrever em documentos PDF. Infelizmente, ler textos de documentos PDF nem sempre resultará em uma tradução perfeita para uma string por causa do formato complicado dos arquivos PDF; alguns PDFs podem até nem ser

legíveis. Nesses casos, você estará sem sorte, a menos que atualizações futuras no PyPDF2 suportem recursos adicionais para os PDFs.

Os documentos Word são mais confiáveis, e podemos lê-los com o módulo python-docx. O texto de documentos Word pode ser manipulado por meio de objetos Paragraph e Run. Esses objetos também podem receber estilos, embora esses estilos devam fazer parte do conjunto default ou ser estilos que já estejam no documento. Podemos acrescentar novos parágrafos, títulos, quebras de linha e de páginas e imagens no documento, porém apenas no final.

Muitas das limitações associadas à manipulação de documentos PDFs e Word devem-se ao fato de esses formatos terem sido criados para serem exibidos de modo elegante a leitores humanos, e não visando a um parse simples pelos softwares. No próximo capítulo, daremos uma olhada em dois outros formatos comuns para armazenar informações: arquivos JSON e CSV. Esses formatos foram concebidos para serem usados pelos computadores, e você verá que o Python é capaz de trabalhar com esses formatos de maneira muito mais fácil.

Exercícios práticos

1. Um valor de string com o nome do arquivo PFF *não* é passado para a função PyPDF2.PdfFileReader(). O que é passado para essa função em seu lugar?
2. Em quais modos os objetos File para PdfFileReader() e para PdfFileWriter() devem ser abertos?
3. Como podemos adquirir um objeto Page para a página 5 de um objeto PdfFileReader?
4. Qual variável de PdfFileReader armazena o número de páginas do documento PDF?
5. Se o PDF de um objeto PdfFileReader estiver criptografado com a senha swordfish, o que devemos fazer antes de podermos obter objetos Page desse PDF?
6. Quais métodos devemos usar para fazer a rotação de uma página?

7. Qual método retorna um objeto Document para um arquivo chamado *demo.docx*?
8. Qual é a diferença entre um objeto Paragraph e um objeto Run?
9. Como podemos obter uma lista de objetos Paragraph para um objeto Document armazenado em uma variável chamada doc?
10. Que tipo de objeto tem variáveis bold, underline, italic, strike e outline?
11. Qual é a diferença entre definir a variável bold com True, False ou None?
12. Como podemos criar um objeto Document para um novo documento Word?
13. Como podemos acrescentar um parágrafo com o texto 'Hello there!' em um objeto Document armazenado em uma variável chamada doc?
14. Quais inteiros representam os níveis de títulos disponíveis em documentos Word?

Projetos práticos

Para exercitar, escreva programas que façam as tarefas a seguir.

Paranoia com PDFs

Usando a função os.walk() do capítulo 9, crie um script que percorra todos os PDFs de uma pasta (e de suas subpastas) e criptografe-os usando uma senha fornecida na linha de comando. Salve cada PDF criptografado com um sufixo *_encrypted.pdf* acrescentado ao nome original do arquivo. Antes de apagar o arquivo original, faça o programa tentar ler e descriptografar o arquivo para garantir que ele tenha sido criptografado corretamente.

Em seguida, escreva um programa que encontre todos os PDFs criptografados em uma pasta (e em suas subpastas) e crie uma cópia descriptografada do PDF usando uma senha especificada. Se a senha estiver incorreta, o programa deverá exibir uma mensagem ao usuário e prosseguir para o próximo PDF.

Convites personalizados como documentos Word

Suponha que você tenha um arquivo-texto com nomes de convidados. Esse

arquivo *guests.txt* contém um nome por linha, da seguinte maneira:

```
Prof. Plum
Miss Scarlet
Col. Mustard
Al Sweigart
RoboCop
```

Crie um programa que gere um documento Word com convites personalizados, conforme mostrado na figura 13.11.

Como o Python-Docx é capaz de usar somente os estilos que já existem no documento Word, será necessário inicialmente adicionar esses estilos em um arquivo Word em branco e então abrir esse arquivo com o Python-Docx. Deve haver um convite por página no documento Word resultante, portanto chame add_break() para adicionar uma quebra de página após o último parágrafo de cada convite. Dessa maneira, será necessário abrir somente um documento Word para imprimir todos os convites de uma só vez.

Figura 13.11 – O documento Word gerado pelo seu script de convites personalizados.

Você pode fazer download de um arquivo *guests.txt* de exemplo a partir de *http://nostarch.com/automatestuff/*.

Programa para quebra de senha de PDF baseado em força

bruta

Suponha que você tenha um PDF criptografado cuja senha tenha esquecido, porém você se recorda que ela era composta de uma única palavra em inglês. Tentar adivinhar sua senha esquecida é uma tarefa bem maçante. Em vez de fazer isso, você pode criar um programa que descriptografará o PDF ao tentar usar todas as palavras possíveis em inglês até que encontre uma que funcione. Esse processo é chamado de *ataque de senha baseado em força bruta*. Faça download do arquivo-texto *dictionary.txt* a partir de *http://nostarch.com/automatestuff/*. Esse *arquivo de dicionário* contém mais de 44 mil palavras em inglês, com uma palavra por linha.

Usando as habilidades para leitura de arquivo adquiridas no capítulo 8, crie uma lista de strings com as palavras lidas desse arquivo. Em seguida, percorra todas as palavras dessa lista em um loop, passando-as ao método `decrypt()`. Se esse método retornar um inteiro igual a 0, a senha estará incorreta e seu programa deverá prosseguir para a próxima senha. Se `decrypt()` retornar `1`, seu programa deverá sair do loop e exibir a senha descoberta. Você deve testar tanto a forma com letras minúsculas quanto a forma com letras maiúsculas de cada palavra. (Em meu laptop, percorrer todas as 88 mil palavras com letras minúsculas e maiúsculas do arquivo de dicionário demorou alguns minutos. É por isso que você não deve usar uma única palavra do dicionário para suas senhas.)

CAPÍTULO 14

TRABALHANDO COM ARQUIVOS CSV E DADOS JSON

No capítulo 13, aprendemos a extrair textos de documentos PDF e Word. Esses arquivos estavam em formato binário, o que exigiu módulos especiais do Python para acessar seus dados. Os arquivos CSV e JSON, por outro lado, são apenas arquivos em formato texto simples. Podemos visualizá-los em um editor de texto, por exemplo, o editor de arquivo do IDLE. Porém o Python também vem com os módulos especiais csv e json, que oferecem funções para ajudar a trabalhar com esses formatos de arquivo.

CSV quer dizer "comma-separated values" (valores separados por vírgula), e os arquivos CSV são planilhas simplificadas armazenadas em arquivos-texto simples. O módulo csv do Python facilita o parse de arquivos CSV.

O JSON (pronuncia-se do mesmo modo que o nome "Jason" em inglês – mas não importa como você pronuncie, pois, de qualquer modo, as pessoas dirão que você está pronunciando incorretamente) é um formato que armazena informações na forma de código-fonte JavaScript em arquivos com formato texto simples. (JSON é a abreviatura de JavaScript Object Notation, ou Notação de Objetos JavaScript). Não é necessário conhecer a linguagem de programação JavaScript para usar arquivos JSON, porém é útil conhecer esse formato, pois o JSON é usado em diversas aplicações web.

Módulo CSV

Cada linha em um arquivo CSV representa uma linha da planilha, e as vírgulas separam as células de uma linha. Por exemplo, a planilha *example.xlsx* de *http://nostarch.com/automatestuff/* terá o seguinte aspecto em um arquivo CSV:

```
4/5/2015 13:34,Apples,73
4/5/2015 3:41,Cherries,85
4/6/2015 12:46,Pears,14
```

```
4/8/2015 8:59,Oranges,52
4/10/2015 2:07,Apples,152
4/10/2015 18:10,Bananas,23
4/10/2015 2:40,Strawberries,98
```

Usarei esse arquivo nos exemplos com o shell interativo neste capítulo. Você pode fazer download de *example.csv* a partir de *http://nostarch.com/automatestuff/* ou inserir o texto em um editor e salvá-lo como *example.csv*.

Os arquivos CSV são simples e carecem de muitos dos recursos presentes em uma planilha Excel. Por exemplo, os arquivos CSV:

- não têm tipos para seus valores – tudo é string;
- não têm configurações para tamanho de fonte nem cor;
- não permitem ter várias planilhas;
- não permitem especificar larguras e alturas das células;
- não permitem mesclar células;
- não permitem incluir imagens ou gráficos.

A vantagem dos arquivos CSV está na simplicidade. Os arquivos CSV são amplamente suportados por diversos tipos de programa, podem ser visualizados em editores de texto (incluindo o editor de arquivo do IDLE) e constituem uma maneira simples de representar dados de planilha. O formato CSV é exatamente como ele se define: apenas um arquivo-texto com valores separados por vírgula.

Como os arquivos CSV são apenas arquivos-texto, você poderia se sentir tentado a lê-los como uma string e processar essas strings usando as técnicas aprendidas no capítulo 8. Por exemplo, pelo fato de cada célula em um arquivo CSV ser separada com uma vírgula, você poderia simplesmente chamar o método `split()` em cada linha de texto para obter os valores. Porém nem todas as vírgulas em um arquivo CSV representam uma fronteira entre duas células. Os arquivos CSV também têm seu próprio conjunto de caracteres de escape para permitir que vírgulas e outros caracteres sejam incluídos como *parte dos valores*. O método `split()` não trata esses caracteres de escape. Por causa dessas armadilhas em potencial, sempre utilize o módulo `csv` para ler e escrever em arquivos CSV.

Objetos Reader

Para ler dados de um arquivo CSV usando o módulo csv, será preciso criar um objeto Reader. Um objeto Reader permite fazer uma iteração pelas linhas do arquivo CSV. Digite o seguinte no shell interativo, com *example.csv* no diretório de trabalho atual:

```
u >>> import csv
v >>> exampleFile = open('example.csv')
w >>> exampleReader = csv.reader(exampleFile)
x >>> exampleData = list(exampleReader)
y >>> exampleData
  [['4/5/2015 13:34', 'Apples', '73'], ['4/5/2015 3:41', 'Cherries', '85'], ['4/6/2015 12:46',
  'Pears', '14'], ['4/8/2015 8:59', 'Oranges', '52'], ['4/10/2015 2:07', 'Apples', '152'],
  ['4/10/2015 18:10', 'Bananas', '23'], ['4/10/2015 2:40', 'Strawberries', '98']]
```

O módulo csv acompanha o Python, portanto podemos importá-lo u sem que seja necessário instalá-lo antes.

Para ler um arquivo CSV com o módulo csv, inicialmente abra esse arquivo usando a função open() v, como você faria com qualquer outro arquivo-texto. Contudo, em vez de chamar o método read() ou readlines() no objeto File retornado por open(), passe-o para a função csv.reader() w. Isso fará um objeto Reader ser retornado para que você possa usá-lo. Observe que não passamos diretamente uma string com o nome do arquivo à função csv.reader().

A maneira mais direta de acessar os valores do objeto Reader é convertê-los em uma lista Python simples passando esse objeto para list() x. Usar list() nesse objeto Reader fará uma lista de listas ser retornada, que poderá ser armazenada em uma variável como exampleData. Fornecer exampleData no shell fará a lista de listas ser exibida y.

Agora que temos o arquivo CSV como uma lista de listas, podemos acessar o valor de uma linha e de uma coluna específicas usando a expressão exampleData[row][col], em que row é o índice de uma das listas em exampleData e col é o índice do item desejado nessa lista. Digite o seguinte no shell interativo:

```
>>> exampleData[0][0]
'4/5/2015 13:34'
>>> exampleData[0][1]
```

```
'Apples'
>>> exampleData[0][2]
'73'
>>> exampleData[1][1]
'Cherries'
>>> exampleData[6][1]
'Strawberries'
```

exampleData[0][0] acessa a primeira lista e retorna a primeira string, exampleData[0][2] acessa a primeira lista e retorna a terceira string e assim por diante.

Lendo dados de objetos Reader em um loop for

Para arquivos CSV grandes, utilize o objeto Reader em um loop for. Isso evita a necessidade de carregar o arquivo inteiro na memória de uma só vez. Por exemplo, digite o seguinte no shell interativo:

```
>>> import csv
>>> exampleFile = open('example.csv')
>>> exampleReader = csv.reader(exampleFile)
>>> for row in exampleReader:
      print('Row #' + str(exampleReader.line_num) + ' ' + str(row))

Row #1 ['4/5/2015 13:34', 'Apples', '73']
Row #2 ['4/5/2015 3:41', 'Cherries', '85']
Row #3 ['4/6/2015 12:46', 'Pears', '14']
Row #4 ['4/8/2015 8:59', 'Oranges', '52']
Row #5 ['4/10/2015 2:07', 'Apples', '152']
Row #6 ['4/10/2015 18:10', 'Bananas', '23']
Row #7 ['4/10/2015 2:40', 'Strawberries', '98']
```

Após ter importado o módulo csv e criado um objeto Reader a partir do arquivo CSV, podemos percorrer as linhas do objeto Reader em um loop. Cada linha corresponde a uma lista de valores, em que cada valor representa uma célula.

A chamada à função print() exibe o número da linha atual e o conteúdo dessa linha. Para obter o número da linha, utilize a variável line_num do objeto Reader, que contém o número da linha atual.

O objeto Reader pode ser percorrido em um loop somente uma vez. Para reler o arquivo CSV, você deverá chamar csv.reader e criar um objeto Reader.

Objetos Writer

Um objeto Writer permite escrever dados em um arquivo CSV. Para criar um objeto Writer, utilize a função csv.writer(). Digite o seguinte no shell interativo:

```
  >>> import csv
u >>> outputFile = open('output.csv', 'w', newline='')
v >>> outputWriter = csv.writer(outputFile)
  >>> outputWriter.writerow(['spam', 'eggs', 'bacon', 'ham'])
  21
  >>> outputWriter.writerow(['Hello, world!', 'eggs', 'bacon', 'ham'])
  32
  >>> outputWriter.writerow([1, 2, 3.141592, 4])
  16
  >>> outputFile.close()
```

Inicialmente, chame open() e passe 'w' para abrir um arquivo em modo de escrita u. Isso criará o objeto que poderá então ser passado para csv.writer() v a fim de criar um objeto Writer.

No Windows, também será preciso passar uma string vazia para o argumento nomeado newline da função open(). Por razões técnicas que estão além do escopo deste livro, se você se esquecer de definir o argumento newline, as linhas em *output.csv* terão espaçamento duplo, conforme mostrado na figura 14.1.

O método writerow() dos objetos Writer aceita um argumento do tipo lista. Cada valor da lista é inserido em sua própria célula no arquivo CSV de saída. O valor de retorno de writerow() é o número de caracteres escrito no arquivo para essa linha (incluindo os caracteres de quebra de linha).

	A	B	C	D	E	F	G
1	42	2	3	4	5	6	7
2							
3	2	4	6	8	10	12	14
4							
5	3	6	9	12	15	18	21
6							
7	4	8	12	16	20	24	28
8							
9	5	10	15	20	25	30	35
10							

Figura 14.1 – Se você se esquecer do argumento nomeado newline='' *em* open()*, o arquivo CSV terá espaçamento duplo.*

Esse código gera um arquivo *output.csv* com o seguinte aspecto:

```
spam,eggs,bacon,ham
"Hello, world!",eggs,bacon,ham
1,2,3.141592,4
```

Observe como o objeto Writer escapa automaticamente a vírgula no valor 'Hello, world!' com aspas duplas no arquivo CSV. O módulo csv evita que você tenha de lidar com esses casos especiais por conta própria.

Argumentos nomeados delimiter e lineterminator

Suponha que você queira separar as células com um caractere de tabulação no lugar de uma vírgula e queira que as linhas tenham espaçamento duplo. Você pode digitar algo como o seguinte no shell interativo:

```
  >>> import csv
  >>> csvFile = open('example.tsv', 'w', newline='')
u >>> csvWriter = csv.writer(csvFile, delimiter='\t', lineterminator='\n\n')
  >>> csvWriter.writerow(['apples', 'oranges', 'grapes'])
  24
  >>> csvWriter.writerow(['eggs', 'bacon', 'ham'])
  17
  >>> csvWriter.writerow(['spam', 'spam', 'spam', 'spam', 'spam', 'spam'])
  32
  >>> csvFile.close()
```

Essas instruções alteram os caracteres delimitadores e de finalização de

linha em seu arquivo. O *delimitador* (delimiter) é o caractere que aparece entre as células em uma linha. Por padrão, o delimitador em um arquivo CSV é uma vírgula. O *finalizador de linha* (line terminator) é o caractere inserido no final de uma linha. Por padrão, o finalizador de linha é um caractere de quebra de linha. Podemos alterar esses caracteres para valores diferentes usando os argumentos nomeados delimiter e lineterminator em csv.writer().

Passar delimeter='\t' e lineterminator='\n\n' u altera o caractere entre as células para uma tabulação e o caractere entre as linhas para duas quebras de linha. Chamamos então writerow() três vezes para termos três linhas.

Essas instruções geram um arquivo chamado *example.tsv* com o conteúdo a seguir:

```
apples  oranges grapes

eggs    bacon   ham

spam    spam    spam    spam    spam    spam
```

Agora que nossas células estão separadas com tabulações, usamos a extensão de arquivo *.tsv*, indicando valores separados por tabulação (tab-separated values).

Projeto: Removendo o cabeçalho de arquivos CSV

Suponha que você tenha a tarefa maçante de remover a primeira linha de várias centenas de arquivos CSV. Talvez você vá fornecê-los a um processo automatizado que exija somente os dados, e não os cabeçalhos na parte superior das colunas. Você *poderia* abrir cada arquivo no Excel, apagar a primeira linha e salvar o arquivo novamente – mas isso demoraria horas. Vamos criar um programa para fazer isso.

O programa deverá abrir todos os arquivos com extensão *.csv* no diretório de trabalho atual, ler o conteúdo do arquivo CSV e reescrevê-lo sem a primeira linha em um arquivo de mesmo nome. Isso fará o conteúdo antigo do arquivo CSV ser substituído pelo novo conteúdo, sem cabeçalhos.

AVISO Como sempre, quando criar um programa que modifique arquivos, não se esqueça de fazer backup dos arquivos antes, somente para o caso de seu programa

não funcionar da maneira esperada. Você não vai querer apagar acidentalmente seus arquivos originais.

Em geral, seu programa deverá fazer o seguinte:

- Encontrar todos os arquivos CSV no diretório de trabalho atual.
- Ler o conteúdo completo de cada arquivo.
- Gravar o conteúdo, exceto a primeira linha, em um novo arquivo CSV.

No nível do código, isso significa que o programa deverá fazer o seguinte:

- Percorrer uma lista de arquivos com os.listdir() em um loop ignorando os arquivos que não sejam CSV.
- Criar um objeto CSV Reader e ler o conteúdo do arquivo usando o atributo line_num para determinar a linha a ser ignorada.
- Criar um objeto CSV Writer e gravar os dados lidos no novo arquivo.

Para esse projeto, abra uma nova janela no editor de arquivo e salve o programa como *removeCsvHeader.py*.

Passo 1: Percorrer todos os arquivos CSV em um loop

A primeira tarefa de seu programa será percorrer todos os arquivos CSV do diretório de trabalho atual em um loop. Faça seu programa *removeCsvHeader.py* ter o seguinte aspecto:

```
#! python3
# removeCsvHeader.py – Remove o cabeçalho de todos os arquivos CSV no diretório
# de trabalho atual.

import csv, os

os.makedirs('headerRemoved', exist_ok=True)

# Percorre todos os arquivos no diretório de trabalho atual em um loop.
for csvFilename in os.listdir('.'):
    if not csvFilename.endswith('.csv'):
u       continue    # ignora arquivos que não sejam csv

    print('Removing header from ' + csvFilename + '...')

    # TODO: Lê o arquivo CSV (pula a primeira linha).
```

```
    # TODO: Grava o arquivo CSV.
```

A chamada a os.makedirs() criará uma pasta headerRemoved, em que todos os arquivos CSV sem cabeçalho serão gravados. Um loop for em os.listdir('.') faz a metade do trabalho, porém o loop incluirá *todos* os arquivos do diretório de trabalho, portanto será necessário acrescentar um pouco de código no início do loop para que os nomes de arquivo que não terminem com .csv sejam ignorados. A instrução continue u faz o loop for passar para o próximo nome de arquivo quando um arquivo que não seja CSV for encontrado.

Somente para que haja *alguma* saída à medida que o programa executar, apresente uma mensagem informando em qual arquivo CSV o programa está atuando. Então acrescente alguns comentários TODO para o restante do programa.

Passo 2: Ler o arquivo CSV

O programa não remove a primeira linha do arquivo CSV. Em vez disso, uma nova cópia do arquivo CSV será criada sem a primeira linha. Como o nome do arquivo copiado é igual ao nome do arquivo original, a cópia sobrescreverá o original.

O programa deverá ter uma maneira de controlar se está, nesse momento, na primeira linha do arquivo CSV no loop. Adicione o código a seguir em *removeCsvHeader.py*.

```
#! python3
# removeCsvHeader.py – Remove o cabeçalho de todos os arquivos CSV no diretório
# de trabalho atual.

--trecho removido--

    # Lê o arquivo CSV (pula a primeira linha).
    csvRows = []
    csvFileObj = open(csvFilename)
    readerObj = csv.reader(csvFileObj)
    for row in readerObj:
        if readerObj.line_num == 1:
            continue    # pula a primeira linha
        csvRows.append(row)
```

```
    csvFileObj.close()

    # TODO: Grava o arquivo CSV.
```

O atributo line_num do objeto Reader pode ser usado para determinar qual linha do arquivo CSV está sendo lida no momento. Outro loop for percorrerá as linhas retornadas pelo objeto CSV Reader, e todas as linhas, exceto a primeira, serão adicionadas a csvRows.

À medida que o loop for faz a iteração pelas linhas, o código verifica se readerObj.line_num está definido com 1. Em caso afirmativo, o programa executará um continue para prosseguir para a próxima linha sem acrescentar essa linha a csvRows. Para todas as demais linhas, a condição sempre será False e a linha será adicionada a csvRows.

Passo 3: Gravar o arquivo CSV sem a primeira linha

Agora que csvRows contém todas as linhas, exceto a primeira, a lista deve ser gravada em um arquivo CSV na pasta *headerRemoved*. Adicione o código a seguir em *removeCsvHeader.py*:

```
#! python3
# removeCsvHeader.py – Remove o cabeçalho de todos os arquivos CSV no diretório
# de trabalho atual.
--trecho removido--

# Percorre todos os arquivos no diretório de trabalho atual em um loop.
u for csvFilename in os.listdir('.'):
    if not csvFilename.endswith('.csv'):
        continue    # ignora arquivos que não sejam csv

    --trecho removido--

    # Grava o arquivo CSV.
    csvFileObj = open(os.path.join('headerRemoved', csvFilename), 'w',  newline='')
    csvWriter = csv.writer(csvFileObj)
    for row in csvRows:
        csvWriter.writerow(row)
    csvFileObj.close()
```

O objeto CSV Writer gravará a lista em um arquivo CSV em headerRemoved

usando csvFilename (que usamos também no reader CSV). Isso fará o arquivo original ser sobrescrito.

Após termos criado o objeto Writer, percorremos as sublistas armazenadas em csvRows e gravamos cada uma delas no arquivo.

Depois que o código for executado, o loop for externo u passará para o próximo nome de arquivo em os.listdir('.'). Quando o loop for concluído, o programa terminará.

Para testar seu programa, faça download de *removeCsvHeader.zip* a partir de *http://nostarch.com/automatestuff/* e descompacte-o em uma pasta. Execute o programa *removeCsvHeader.py* nessa pasta. A saída será semelhante a:

```
Removing header from NAICS_data_1048.csv...
Removing header from NAICS_data_1218.csv...
--trecho removido--
Removing header from NAICS_data_9834.csv...
Removing header from NAICS_data_9986.csv...
```

Esse programa deve exibir um nome de arquivo sempre que remover a primeira linha de um arquivo CSV.

Ideias para programas semelhantes

Os programas que podem ser criados para arquivos CSV são semelhantes àqueles usados em arquivos Excel, pois ambos os arquivos contêm planilhas. Você pode criar programas para fazer o seguinte:

- Comparar dados de linhas diferentes em um arquivo CSV ou entre vários arquivos CSV.
- Copiar dados específicos de um arquivo CSV para um arquivo Excel ou vice-versa.
- Verificar se há dados inválidos ou erros de formatação em arquivos CSV e alertar o usuário sobre esses erros.
- Ler dados de um arquivo CSV como entrada para seus programas Python.

JSON e APIs

O JSON (JavaScript Object Notation, ou Notação de Objetos JavaScript) é uma maneira popular de formatar dados como uma única string legível aos seres humanos. Essa é a maneira nativa com a qual os programas JavaScript definem suas estruturas de dados e, normalmente, lembra o resultado da função pprint() do Python. Não é preciso conhecer JavaScript para trabalhar com dados formatados em JSON.

Eis um exemplo de dados formatados com o JSON:

```
{"name": "Zophie", "isCat": true,
"miceCaught": 0, "napsTaken": 37.5,
"felineIQ": null}
```

É conveniente conhecer o JSON, pois muitos sites oferecem conteúdo JSON como uma maneira de os programas interagirem com o site. Isso é conhecido como prover uma *API* (*Application Programming Interface*, ou Interface de Programação de Aplicativos). Acessar uma API é o mesmo que acessar qualquer página web por meio de um URL. A diferença está no fato de os dados retornados por uma API serem formatados para os computadores (com JSON, por exemplo); não é fácil para as pessoas lerem as APIs.

Muitos sites disponibilizam seus dados em formato JSON. Facebook, Twitter, Yahoo, Google, Tumblr, Wikipedia, Flickr, Data.gov, Reddit, IMDb, Rotten Tomatoes, LinkedIn e vários outros sites populares oferecem APIs para os programas usarem. Alguns desses sites exigem uma inscrição, que quase sempre é gratuita. Será necessário localizar a documentação que explique para quais URLs seu programa deve fazer uma solicitação a fim de obter os dados desejados, assim como para ver o formato geral das estruturas de dados JSON retornado. Essa documentação deve ser fornecida por qualquer site que ofereça a API; se houver uma página "Developers" (Desenvolvedores), procure a documentação nesse local.

Ao usar APIs, você pode criar programas para fazer o seguinte:

- Extrair dados puros dos sites. (Acessar APIs, em geral, é mais conveniente do que fazer download de páginas web e fazer parse do HTML com o Beautiful Soup.)
- Fazer download automaticamente de novas postagens de uma de suas contas de redes sociais e postá-las em outra conta. Por exemplo, você pode obter

seus posts do Tumblr e colocá-los no Facebook.

- Criar uma "enciclopédia de filmes" para sua coleção pessoal de filmes ao extrair dados do IMDb, do Rotten Tomatoes e da Wikipedia e colocá-los em um único arquivo-texto em seu computador.

Você pode ver alguns exemplos de APIs JSON nos recursos em *http://nostarch.com/automatestuff/*.

Módulo json

O módulo json do Python cuida de todos os detalhes da tradução entre uma string com dados JSON e valores Python para as funções json.loads() e json.dumps(). O JSON não é capaz de armazenar *todos* os tipos de valores Python. Ele pode conter valores somente dos seguintes tipos de dados: strings, inteiros, pontos flutuantes, booleanos, listas, dicionários e NoneType. O JSON não é capaz de representar objetos específicos do Python como objetos File, objetos CSV Reader ou Writer, objetos Regex ou objetos WebElement do Selenium.

Lendo JSON com a função loads()

Para traduzir uma string contendo dados JSON em um valor Python, passe-a para a função json.loads(). [O nome quer dizer "load string" (carregar string), e não "loads" como verbo em inglês na terceira pessoa do singular.] Digite o seguinte no shell interativo:

```
>>> stringOfJsonData = '{"name": "Zophie", "isCat": true, "miceCaught": 0,
"felineIQ": null}'
>>> import json
>>> jsonDataAsPythonValue = json.loads(stringOfJsonData)
>>> jsonDataAsPythonValue
{'isCat': True, 'miceCaught': 0, 'name': 'Zophie', 'felineIQ': None}
```

Após importar o módulo json, podemos chamar loads() e passar-lhe uma string com dados JSON. Observe que as strings JSON sempre utilizam aspas duplas. Esses dados serão retornados na forma de um dicionário Python. Os dicionários Python não são ordenados, portanto os pares chave-valor poderão aparecer em uma ordem diferente quando jsonDataAsPythonValue for exibido.

Escrevendo JSON com a função dumps()

A função json.dumps() (que quer dizer “dump string”, e não “dumps” como verbo em inglês na terceira pessoa do singular) traduzirá um valor Python em uma string de dados formatada em JSON. Digite o seguinte no shell interativo:

```
>>> pythonValue = {'isCat': True, 'miceCaught': 0, 'name': 'Zophie','felineIQ':
None}
>>> import json
>>> stringOfJsonData = json.dumps(pythonValue)
>>> stringOfJsonData
'{"isCat": true, "felineIQ": null, "miceCaught": 0, "name": "Zophie" }'
```

O valor pode corresponder somente a um dos seguintes tipos de dados básicos do Python: dicionário, lista, inteiro, ponto flutuante, string, booleano ou None.

Projeto: Acessando dados atuais de previsão do tempo

Verificar a previsão do tempo parece uma tarefa bastante trivial: abra seu navegador web, clique na barra de endereço, digite o URL de um site de previsão do tempo (ou procure um e, em seguida, clique no link), espere a página ser carregada, ignore as propagandas, e assim por diante.

Na realidade, há vários passos maçantes que poderão ser pulados se você tiver um programa que faça o download da previsão do tempo para os próximos dias e apresente essas informações em formato texto simples. Esse programa utiliza o módulo requests do capítulo 11 para fazer download dos dados da Web.

Em geral, o programa deve fazer o seguinte:

- Ler a localidade solicitada a partir da linha de comando.
- Fazer download dos dados JSON de previsão de tempo de OpenWeatherMap.org.
- Converter a string com dados JSON em uma estrutura de dados Python.
- Exibir a previsão do tempo para hoje e para os próximos dois dias.

Sendo assim, o código deverá fazer o seguinte:

- Unir strings em sys.argv para obter a localidade.
- Chamar requests.get() para fazer download dos dados da previsão do tempo.
- Chamar json.loads() para converter os dados JSON em uma estrutura de dados Python.
- Exibir a previsão do tempo.

Para esse projeto, abra uma nova janela no editor de arquivo e salve o programa como *quickWeather.py*.

Passo 1: Obter a localidade a partir dos argumentos da linha de comando

A entrada para esse programa será obtida da linha de comando. Faça *quickWeather.py* ter o aspecto a seguir:

```
#! python3
# quickWeather.py – Exibe a previsão do tempo para uma localidade obtida na linha de
comando.

import json, requests, sys

# Processa a localidade a partir dos argumentos da linha de comando.
if len(sys.argv) < 2:
    print('Usage: quickWeather.py location')
    sys.exit()
location = ' '.join(sys.argv[1:])

# TODO: Faz download dos dados JSON a partir da API de OpenWeatherMap.org.

# TODO: Carrega dados JSON em uma variável Python.
```

Em Python, os argumentos da linha de comando são armazenados na lista sys.argv. Após a linha shebang #! e as instruções import, o programa verifica se há mais de um argumento na linha de comando. (Lembre-se de que sys.argv sempre terá pelo menos um elemento sys.argv[0] que contém o nome do arquivo de script Python.) Se houver apenas um elemento na lista, o usuário não especificou uma localidade na linha de comando e uma mensagem de “uso” será fornecida ao usuário antes de o programa ser encerrado.

Os argumentos da linha de comando são separados por espaços. O argumento de linha de comando San Francisco, CA fará sys.argv armazenar ['quickWeather.py', 'San', 'Francisco,', 'CA']. Sendo assim, chame o método join() para unir todas as strings em sys.argv, exceto a primeira. Armazene essa string resultante dessa união em uma variável chamada location.

Passo 2: Fazer download dos dados JSON

OpenWeatherMap.org provê informações de previsão do tempo em tempo real, em formato JSON. Seu programa simplesmente deve fazer download da página em *http://api.openweathermap.org/data/2.5/forecast/daily?q=<Localidade>&cnt=3*, em que *<Localidade>* é o nome da cidade cuja previsão do tempo você quer obter. Adicione o código a seguir em *quickWeather.py*.

```
#! python3
# quickWeather.py – Exibe a previsão do tempo para uma localidade obtida na linha de
comando.

--trecho removido--

# Faz download dos dados JSON a partir da API de OpenWeatherMap.org.
url ='http://api.openweathermap.org/data/2.5/forecast/daily?q=%s&cnt=3' %
(location)
response = requests.get(url)
response.raise_for_status()

# TODO: Carrega dados JSON em uma variável Python.
```

Temos location, obtido a partir dos argumentos da linha de comando. Para compor o URL que queremos acessar, utilizamos o placeholder %s e inserimos a string armazenada em location nessa posição da string de URL. Armazenamos o resultado em url e o passamos para requests.get(). A chamada a requests.get() retorna um objeto Response; podemos verificar se houve algum erro chamando raise_for_status(). Se nenhuma exceção for gerada, o texto baixado estará em response.text.

Passo 3: Carregar dados JSON e exibir informações

sobre a previsão do tempo

A variável response.text armazena uma string longa contendo dados formatados em JSON. Para converter esses dados em um valor Python, chame a função json.loads(). Os dados JSON terão uma aparência semelhante a:

```
{'city': {'coord': {'lat': 37.7771, 'lon': -122.42},
          'country': 'United States of America',
          'id': '5391959',
          'name': 'San Francisco',
          'population': 0},
 'cnt': 3,
 'cod': '200',
 'list': [{'clouds': 0,
           'deg': 233,
           'dt': 1402344000,
           'humidity': 58,
           'pressure': 1012.23,
           'speed': 1.96,
           'temp': {'day': 302.29,
                    'eve': 296.46,
                    'max': 302.29,
                    'min': 289.77,
                    'morn': 294.59,
                    'night': 289.77},
           'weather': [{'description': 'sky is clear',
                        'icon': '01d',
--trecho removido--
```

Podemos ver esses dados se passarmos weatherData a pprint.pprint(). Você pode dar uma olhada em *http://openweathermap.org/* para obter uma documentação adicional sobre o que esses campos significam. Por exemplo, a documentação online informará que 302.29 após 'day' é a temperatura do dia em Kelvin, e não em Celsius ou Fahrenheit.

As descrições da previsão do tempo desejadas estão após 'main' e 'description'. Para exibi-las de modo elegante, acrescente o código a seguir em *quickWeather.py*.

```
! python3
```

```
# quickWeather.py – Exibe a previsão do tempo para uma localidade obtida na linha de
comando.

--trecho removido--

# Carrega dados JSON em uma variável Python.
weatherData = json.loads(response.text)

# Exibe as descrições da previsão do tempo.
u w = weatherData['list']
print('Current weather in %s:' % (location))
print(w[0]['weather'][0]['main'], '-', w[0]['weather'][0]['description'])
print()
print('Tomorrow:')
print(w[1]['weather'][0]['main'], '-', w[1]['weather'][0]['description'])
print()
print('Day after tomorrow:')
print(w[2]['weather'][0]['main'], '-', w[2]['weather'][0]['description'])
```

Observe como o código armazena weatherData['list'] na variável w para economizar algumas digitações u. Utilize w[0], w[1] e w[2] para obter os dicionários com a previsão do tempo para hoje, amanhã e depois de amanhã, respectivamente. Cada um desses dicionários tem uma chave 'weather' que contém um valor de lista. Você está interessado no primeiro item da lista no índice 0, que é um dicionário aninhado com várias chaves. Nesse caso, exibimos os valores armazenados nas chaves 'main' e 'description', separados por um hífen.

Quando esse programa for executado com o argumento de linha de comando quickWeather.py San Francisco, CA, a saída será semelhante a:

```
Current weather in San Francisco, CA:
Clear - sky is clear

Tomorrow:
Clouds - few clouds

Day after tomorrow:
Clear - sky is clear
```

(O clima é um dos motivos pelos quais eu moro em San Francisco!).

Ideias para programas semelhantes

Acessar dados de previsão do tempo pode constituir a base de diversos tipos de programas. Podemos criar programas semelhantes para fazer o seguinte:

- Obter previsões do tempo para diversos pontos de camping ou trilhas para caminhada e ver em qual deles o clima estará mais agradável.
- Agendar um programa que verifique regularmente a previsão do tempo e envie um alerta de geada se for necessário colocar suas plantas para dentro de casa. (O capítulo 15 discute o agendamento e o capítulo 16 explica como podemos enviar emails.)
- Extrair dados de previsão do tempo de vários sites para mostrá-los de uma só vez ou calcular e mostrar a média de várias previsões do tempo.

Resumo

CSV e JSON são formatos comuns de texto simples para armazenar dados. É fácil para os programas fazerem parse desses dados, ao mesmo tempo que eles são legíveis aos seres humanos; sendo assim, esses dados são usados em planilhas simples ou como dados de aplicações web. Os módulos csv e json simplificam bastante o processo de leitura e de escrita de arquivos CSV e JSON.

Os últimos capítulos ensinaram a usar o Python para fazer parse de informações em uma ampla variedade de formatos de arquivos. Uma tarefa comum consiste em obter dados de uma variedade de formatos e fazer seu parse em busca das informações necessárias em particular. Essas tarefas geralmente são específicas a ponto de os softwares comerciais não serem convenientes. Ao criar seus próprios scripts, você poderá fazer o computador manipular grandes quantidades de dados apresentados nesses formatos.

No capítulo 15, deixaremos os formatos de dados de lado e aprenderemos a criar programas que se comuniquem com você enviando emails e mensagens de texto.

Exercícios práticos

1. Quais são alguns recursos presentes em planilhas Excel que não estão em planilhas CSV?
2. O que devemos passar para csv.reader() e para csv.writer() para criar objetos Reader e Writer?
3. Em quais modos os objetos File para objetos Reader e Writer devem ser abertos?
4. Qual método aceita um argumento de lista e o grava em um arquivo CSV?
5. O que os argumentos nomeados delimiter e lineterminator fazem?
6. Qual função aceita uma string com dados JSON e retorna uma estrutura de dados Python?
7. Qual função aceita uma estrutura de dados Python e retorna uma string com dados JSON?

Projeto prático

Para exercitar, escreva um programa que execute a tarefa a seguir.

Conversor de Excel para CSV

O Excel pode salvar uma planilha em um arquivo CSV com alguns cliques de mouse, porém, se for preciso converter centenas de arquivos Excel em CSVs, isso exigirá horas clicando. Usando o módulo openpyxl do capítulo 12, crie um programa que leia todos os arquivos Excel do diretório de trabalho atual e gere arquivos CSV.

Um único arquivo Excel pode conter diversas planilhas; você deverá criar um arquivo CSV por *planilha*. Os nomes dos arquivos CSV devem ser *<nome do arquivo excel>_<título da planilha>.csv*, em que *<nome do arquivo excel>* é o nome do arquivo Excel sem a extensão (por exemplo, 'spam_data', e não 'spam_data.xlsx') e *<título da planilha>* é a string presente na variável title do objeto Worksheet.

Esse programa envolverá muitos loops for aninhados. O esqueleto do programa é semelhante a:

```
for excelFile in os.listdir('.'):
    # Ignora os arquivos que não sejam xlsx, carrega o objeto workbook.
```

```
    for sheetName in wb.get_sheet_names():
        # Percorre todas as planilhas do workbook em um loop.
        sheet = wb.get_sheet_by_name(sheetName)
        # Cria o nome do arquivo CSV a partir do nome do arquivo Excel e do título da
planilha.
        # Cria o objeto csv.writer para esse arquivo CSV.

        # Percorre todas as linhas da planilha em um loop.
        for rowNum in range(1, sheet.get_highest_row() + 1):
            rowData = []    # adiciona todas as células a essa lista
            # Percorre todas as células da linha em um loop.
            for colNum in range(1, sheet.get_highest_column() + 1):
                # Adiciona os dados de cada célula em rowData.

            # Grava a lista rowData no arquivo CSV.

        csvFile.close()
```

Faça download do arquivo ZIP *excelSpreadsheets.zip* a partir de *http://nostarch.com/automatestuff/* e descompacte as planilhas no mesmo diretório em que estiver o seu programa. Você pode usar esses arquivos para testá-lo.

CAPÍTULO 15

MONITORANDO TEMPO, AGENDANDO TAREFAS E INICIANDO PROGRAMAS

Executar programas enquanto você está sentado diante de seu computador é bom, porém fazer os programas serem executados sem a sua supervisão direta também é conveniente. O relógio de seu computador pode agendar programas para executar código em alguma data e hora especificadas ou a intervalos regulares. Por exemplo, seu programa pode extrair informações de um site a cada hora para verificar se houve alterações ou pode realizar uma tarefa que exija bastante CPU às quatro horas da manhã enquanto você estiver dormindo. Os módulos time e datetime do Python disponibilizam essas funções.

Também podemos criar programas que iniciem outros programas de acordo com uma agenda usando os módulos subprocess e threading. Geralmente, tirar proveito de aplicações que outras pessoas já escreveram é a maneira mais rápida de programar.

Módulo time

O relógio de sistema de seu computador está definido com uma data, uma hora e um fuso horário específicos. O módulo pronto time permite que seus programas Python leiam o relógio do sistema para obter o horário atual. As funções time.time() e time.sleep() são as mais úteis desse módulo.

Função time.time()

O *Unix epoch* (Era Unix ou Época Unix) é uma referência de tempo comumente usada em programação: 00:00 hora de 1 de janeiro de 1970 UTC (Universal Time Coordinated, ou Tempo Universal Coordenado). A função time.time() retorna o número de segundos desde esse momento na forma de um valor de ponto flutuante. (Lembre-se de que um número de ponto flutuante é

somente um número com um ponto decimal.) Esse número é chamado de *epoch imestamp*. Por exemplo, digite o seguinte no shell interativo:

```
>>> import time
>>> time.time()
1425063955.068649
```

Nesse caso, chamei time.time() em 27 de fevereiro de 2015 às 11h05min no fuso Pacific Standard Time, ou seja, às 19h05min UTC. O valor de retorno corresponde à quantidade de segundos transcorridos entre o Unix epoch e o instante em que time.time() foi chamado.

NOTA Os exemplos no shell interativo fornecerão datas e horas referentes ao momento em que escrevi este capítulo em fevereiro de 2015. A menos que você seja um viajante do tempo, suas datas e horas serão diferentes.

Os epoch timestamps podem ser usados para gerar o perfil (*profile*) do código, ou seja, para medir quanto tempo uma porção de código demora para executar. Se time.time() for chamado no início do bloco de código em que você quer fazer a medida e, novamente, no final, você poderá subtrair o primeiro timestamp do segundo e descobrir o tempo decorrido entre essas duas chamadas. Por exemplo, abra uma nova janela no editor de arquivo e digite o programa a seguir:

```
  import time
u def calcProd():
      # Calcula o produto dos 100.000 primeiros números.
      product = 1
      for i in range(1, 100000):
          product = product * i
      return product

v startTime = time.time()
  prod = calcProd()
w endTime = time.time()
x print('The result is %s digits long.' % (len(str(prod))))
y print('Took %s seconds to calculate.' % (endTime - startTime))
```

Em u definimos uma função calcProd() para percorrer os inteiros de 1 a 99.999 em um loop e retornar o seu produto. Em v chamamos time.time() e armazenamos seu valor em startTime. Logo após chamar calcProd(), chamamos

time.time() novamente e armazenamos seu valor em endTime w. Concluímos exibindo o tamanho do produto retornado por calcProd() x e mostrando quanto tempo demorou para que calcProd() fosse executado y.

Salve esse programa como *calcProd.py* e execute-o. A saída será semelhante a:

```
The result is 456569 digits long.
Took 2.844162940979004 seconds to calculate.
```

NOTA Outra maneira de gerar o perfil de seu código consiste em usar a função cProfile.run(), que fornece um nível de detalhes muito mais repleto de informações do que a técnica simples de usar time.time(). A explicação da função cProfile.run() encontra-se em *https://docs.python.org/3/library/profile.html.*

Função time.sleep()

Se for necessário fazer uma pausa em seu programa, chame a função time.sleep() e passe-lhe o número de segundos que você deseja que seu programa permaneça parado. Digite o seguinte no shell interativo:

```
  >>> import time
  >>> for i in range(3):
u         print('Tick')
v         time.sleep(1)
w         print('Tock')
x         time.sleep(1)

  Tick
  Tock
  Tick
  Tock
  Tick
  Tock
y >>> time.sleep(5)
```

O loop for exibirá Tick u, fará uma pausa de um segundo v, exibirá Tock w, fará uma pausa de um segundo x, exibirá Tick, fará uma pausa, e assim sucessivamente, até que Tick e Tock tenham sido exibidos três vezes cada um.

A função time.sleep() ficará *bloqueada* – ou seja, não retornará nem deixará seu programa executar outros códigos – até que o número de segundos

passados para time.sleep() tenha decorrido. Por exemplo, se time.sleep(5) y for especificado, você verá que o próximo prompt (>>>) não aparecerá até que cinco segundos tenham passado.

Saiba que pressionar CTRL-C não interromperá as chamadas a time.sleep() no IDLE. O IDLE espera até que toda a pausa termine para gerar a exceção KeyboardInterrupt. Para contornar esse problema, em vez de ter uma única chamada a time.sleep(30) para fazer uma pausa de 30 segundos, utilize um loop for para fazer 30 chamadas a time.sleep(1).

```
>>> for i in range(30):
    time.sleep(1)
```

Se CTRL-C for pressionado em algum momento durante esses 30 segundos, você verá a exceção KeyboardInterrupt ser gerada imediatamente.

Arredondando números

Ao trabalhar com tempo, frequentemente você encontrará valores de ponto flutuante com vários dígitos após o decimal. Para trabalhar mais facilmente com esses valores, você poderá abreviá-los com a função interna round() do Python, que arredonda um valor de ponto flutuante com a precisão especificada. Basta passar o número que você quer arredondar, além de um segundo argumento opcional que representa para quantos dígitos após o ponto decimal você quer arredondar. Se o segundo argumento for omitido, round() arredondará seu número para o inteiro mais próximo. Digite o seguinte no shell interativo:

```
>>> import time
>>> now = time.time()
>>> now
1425064108.017826
>>> round(now, 2)
1425064108.02
>>> round(now, 4)
1425064108.0178
>>> round(now)
1425064108
```

Após importar time e armazenar time.time() em now, chamamos round(now, 2) para arredondar now com dois dígitos após o decimal, round(now, 4) para arredondar com quatro dígitos após o decimal e round(now) para arredondar para o inteiro mais próximo.

Projeto: Supercronômetro

Suponha que você queira monitorar quanto tempo você gasta em tarefas maçantes que ainda não tenham sido automatizadas. Não é preciso ter um cronômetro físico, e é surpreendentemente difícil encontrar um aplicativo gratuito de cronômetro para seu laptop ou seu smartphone que não esteja repleto de propagandas e que não envie uma cópia do histórico de seu navegador às empresas comerciais. (O contrato de licença com o qual você concordou afirma que isso pode ser feito. Você leu o contrato de licença, não é mesmo?) Podemos criar um programa simples de cronômetro em Python.

De modo geral, eis o que o seu programa deverá fazer:

- Monitorar a quantidade de tempo decorrida entre pressionamentos da tecla ENTER, em que cada pressionamento de tecla iniciará uma nova "rodada" do timer.
- Exibir o número da rodada, o tempo total e o tempo de duração da rodada.

Isso significa que o seu código deverá fazer o seguinte:

- Determinar o horário atual chamando time.time() e armazenar seu valor como um timestamp no início do programa, assim como no início de cada rodada.
- Manter um contador de rodadas e incrementá-lo sempre que o usuário teclar ENTER.
- Calcular o tempo decorrido subtraindo os timestamps.
- Tratar a exceção KeyboardInterrupt para que o usuário possa pressionar CTRL-C para sair.

Abra uma nova janela no editor de arquivo e salve esse arquivo como *stopwatch.py*.

Passo 1: Preparar o programa para monitorar tempos

O programa de cronômetro deverá usar o horário atual, portanto será

necessário importar o módulo time. Seu programa também deverá exibir algumas instruções básicas ao usuário antes de chamar input(), portanto o timer poderá ser iniciado após o usuário teclar ENTER. Em seguida, o código começará a monitorar os tempos de duração das rodadas.

Digite o código a seguir no editor de arquivo, escrevendo um comentário TODO como placeholder para o restante do código:

```
#! python3
# stopwatch.py – Um programa simples de cronômetro.

import time

# Exibe as instruções do programa.
print('Press ENTER to begin. Afterwards, press ENTER to "click" the stopwatch. Press
Ctrl-C to quit.')
input()                  # tecle Enter para começar
print('Started.')
startTime = time.time()    # obtém o horário de início da primeira rodada
lastTime = startTime
lapNum = 1

# TODO: Começa a monitorar a duração das rodadas.
```

Agora que criamos o código para exibir as instruções, inicie a primeira rodada, anote o horário e defina nosso contador de rodadas com 1.

Passo 2: Monitorar e exibir os tempos de duração das rodadas

Vamos agora escrever o código que inicia cada nova rodada, calcula a duração da rodada anterior e o tempo total decorrido desde que o cronômetro foi iniciado. Exibiremos o tempo de duração da rodada e o tempo total, além de incrementarmos o contador de rodadas a cada nova rodada. Acrescente o código a seguir em seu programa:

```
#! python3
# stopwatch.py – Um programa simples de cronômetro.

import time
```

```
   --trecho removido--

   # Começa a monitorar a duração das rodadas.
u try:
v    while True:
         input()
w        lapTime = round(time.time() - lastTime, 2)
x        totalTime = round(time.time() - startTime, 2)
y        print('Lap #%s: %s (%s)' % (lapNum, totalTime, lapTime), end='')
         lapNum += 1
         lastTime = time.time() # reinicia a última rodada
z except KeyboardInterrupt:
     # Trata a exceção de Ctrl-C para evitar que sua mensagem de erro seja exibida.
     print('\nDone.')
```

Se o usuário pressionar CTRL-C para interromper o cronômetro, a exceção KeyboardInterrupt será gerada e o programa falhará se a execução não estiver em uma instrução try. Para evitar a falha, incluímos essa parte do programa em uma instrução try u. Trataremos a exceção na cláusula except z, portanto, quando CTRL-C for pressionado e a exceção for gerada, a execução do programa será desviada para a cláusula except e Done será exibido no lugar da mensagem de erro de KeyboardInterrupt. Até isso acontecer, a execução estará em um loop infinito v que chama input() e espera até o usuário pressionar ENTER para finalizar uma rodada. Quando uma rodada terminar, calcularemos quanto tempo ela demorou subtraindo o instante de início da rodada, ou seja, lastTime, do instante atual, isto é, time.time() w. Calculamos o tempo total decorrido subtraindo o instante de início geral do cronômetro, ou seja, startTime, do instante atual x.

Como os resultados desses cálculos de tempo terão muitos dígitos após o ponto decimal (por exemplo, 4.766272783279419), usamos a função round() para arredondar o valor de ponto flutuante para dois dígitos em w e em x.

Em y exibimos o número da rodada, o tempo total decorrido e a duração da rodada. Como o fato de o usuário teclar ENTER para a chamada a input() faz uma quebra de linha ser exibida na tela, passe end='' à função print() para evitar um espaçamento duplo na saída. Após exibir as informações da rodada, estaremos prontos para a próxima rodada ao somar um ao contador lapNum e

definir lastTime com o instante atual, que será o instante de início da próxima rodada.

Ideias para programas semelhantes

A monitoração de tempo cria diversas possibilidades para seus programas. Embora você possa fazer download de aplicativos que façam algumas dessas tarefas, a vantagem de criar programas por conta própria está no fato de eles serem gratuitos e não estarem repletos de propagandas e de recursos desnecessários. Você pode criar programas semelhantes para fazer o seguinte:

- Criar um aplicativo simples de controle de ponto que faça registros quando você digitar o nome de uma pessoa e que utilize o horário atual para registrar seus horários de entrada e de saída.
- Adicionar um recurso ao seu programa para exibir o tempo decorrido desde o início de um processo, por exemplo, um download que utilize o módulo requests. (Veja o capítulo 11.)
- Verificar intermitentemente há quanto tempo um programa está executando e oferecer ao usuário uma oportunidade para cancelar tarefas que estejam demorando demais.

Módulo datetime

O módulo time é útil para obter um timestamp Unix epoch com o qual trabalhar. Porém, se você quiser exibir uma data em um formato mais conveniente ou realizar operações aritméticas com datas (por exemplo, descobrir qual a data correspondente a 205 dias atrás ou qual será a data dentro de 123 dias), você deverá usar o módulo datetime.

O momento datetime tem seu próprio tipo de dado datetime. Valores do tipo datetime representam um momento específico no tempo. Digite o seguinte no shell interativo:

```
  >>> import datetime
u >>> datetime.datetime.now()
v datetime.datetime(2015, 2, 27, 11, 10, 49, 55, 53)
w >>> dt = datetime.datetime(2015, 10, 21, 16, 29, 0)
x >>> dt.year, dt.month, dt.day
```

```
(2015, 10, 21)
y >>> dt.hour, dt.minute, dt.second
(16, 29, 0)
```

Chamar datetime.datetime.now() u faz um objeto datetime ser retornado v para a data e a hora atuais, de acordo com o relógio de seu computador. Esse objeto inclui o ano, o mês, o dia, a hora, o minuto, o segundo e o microssegundo do instante atual. Podemos também obter um objeto datetime para um instante específico usando a função datetime.datetime() w e passando-lhe inteiros que representem o ano, o mês, o dia, a hora e o segundo do instante desejado. Esses inteiros serão armazenados nos atributos year, month, day x, hour, minute e second y do objeto datetime.

Um timestamp Unix epoch pode ser convertido em um objeto datetime com a função datetime.datetime.fromtimestamp(). A data e a hora do objeto datetime serão convertidas de acordo com o fuso horário local. Digite o seguinte no shell interativo:

```
>>> datetime.datetime.fromtimestamp(1000000)
datetime.datetime(1970, 1, 12, 5, 46, 40)
>>> datetime.datetime.fromtimestamp(time.time())
datetime.datetime(2015, 2, 27, 11, 13, 0, 604980)
```

Chamar datetime.datetime.fromtimestamp() e passar o valor 1000000 faz um objeto datetime para o instante correspondente a 1 milhão de segundos após o Unix epoch ser retornado. Se passarmos time.time(), que é o timestamp Unix epoch do instante atual, um objeto datetime para o instante atual será retornado. Desse modo, as expressões datetime.datetime.now() e datetime.datetime.fromtimestamp(time.time()) fazem o mesmo; ambas retornam um objeto datetime para o instante atual.

NOTA Esses exemplos foram executados em um computador configurado com o fuso horário Pacific Standard Time. Se você estiver em outro fuso horário, seus resultados serão diferentes.

Os objetos datetime podem ser comparados uns aos outros com os operadores de comparação para descobrir qual deles é anterior ao outro. O objeto datetime mais recente terá o valor “maior”. Digite o seguinte no shell interativo:

```
u >>> halloween2015 = datetime.datetime(2015, 10, 31, 0, 0, 0)
v >>> newyears2016 = datetime.datetime(2016, 1, 1, 0, 0, 0)
  >>> oct31_2015 = datetime.datetime(2015, 10, 31, 0, 0, 0)
w >>> halloween2015 == oct31_2015
  True
x >>> halloween2015 > newyears2016
  False
y >>> newyears2016 > halloween2015
  True
  >>> newyears2016 != oct31_2015
  True
```

Crie um objeto datetime para o primeiro instante (meia-noite) do dia 31 de outubro de 2015 e armazene-o em halloween2015 u. Crie um objeto datetime para o primeiro instante de 1 de janeiro de 2016 e armazene-o em newyears2016 v. Em seguida, crie outro objeto para a zero hora do dia 31 de outubro de 2015 e armazene-o em oct31_2015. A comparação entre halloween2015 e oct31_2015 mostra que ambos são iguais w. Comparar newyears2016 e halloween2015 mostra que newyears2016 é maior (mais recente) que halloween2015 x y.

Tipo de dado timedelta

O módulo datetime também disponibiliza um tipo de dado timedelta que representa uma *duração*, em vez de um *instante* no tempo. Digite o seguinte no shell interativo:

```
u >>> delta = datetime.timedelta(days=11, hours=10, minutes=9, seconds=8)
v >>> delta.days, delta.seconds, delta.microseconds
  (11, 36548, 0)
  >>> delta.total_seconds()
  986948.0
  >>> str(delta)
  '11 days, 10:09:08'
```

Para criar um objeto timedelta, utilize a função datetime.timedelta(). A função datetime.timedelta() aceita os argumentos nomeados weeks, days, hours, minutes, seconds, milliseconds e microseconds. Não há nenhum argumento nomeado month ou year, pois “um mês” ou “um ano” corresponde a uma quantidade variável

de tempo de acordo com o mês ou o ano em particular. Um objeto timedelta contém a duração total representada em dias, segundos e microssegundos. Esses números são armazenados nos atributos days, seconds e microseconds, respectivamente. O método total_seconds() retorna a duração somente em número de segundos. Passar um objeto timedelta a str() retornará uma representação em string do objeto, elegantemente formatada e legível aos seres humanos.

Nesse exemplo, passamos argumentos nomeados a datetime.delta() para especificar uma duração de 11 dias, 10 horas, 9 minutos e 8 segundos, e armazenamos o objeto timedelta retornado em delta u. O atributo days desse objeto timedelta armazena 11 e seu atributo seconds armazena 36548 (10 horas, 9 minutos e 8 segundos expressos em segundos) v. Chamar total_seconds() nos informa que 11 dias, 10 horas, 9 minutos e 8 segundos correspondem a 986.948 segundos. Por fim, passar o objeto timedelta a str() retorna uma string que explica claramente a duração.

Os operadores aritméticos podem ser usados para realizar *operações aritméticas com datas* em valores datetime. Por exemplo, para calcular a data correspondente a 1.000 dias a partir de agora, digite o seguinte no shell interativo:

```
>>> dt = datetime.datetime.now()
>>> dt
datetime.datetime(2015, 2, 27, 18, 38, 50, 636181)
>>> thousandDays = datetime.timedelta(days=1000)
>>> dt + thousandDays
datetime.datetime(2017, 11, 23, 18, 38, 50, 636181)
```

Inicialmente, crie um objeto datetime para o instante atual e armazene-o em dt. Em seguida, crie um objeto timedelta para uma duração correspondente a 1.000 dias e armazene-o em thousandDays. Some dt e thousandDays para obter um objeto datetime para a data correspondente a 1.000 dias a partir de agora. O Python fará a operação aritmética com as datas para descobrir que 1.000 dias após o dia 27 de fevereiro de 2015 será o dia 23 de novembro de 2017. Isso é útil porque, ao calcular 1.000 dias a partir de uma data especificada, você deveria se lembrar de quantos dias há em cada mês e considerar anos bissextos e outros detalhes intrincados. O módulo datetime cuida de tudo isso

para você.

Os objetos timedelta podem ser somados ou subtraídos de objetos datetime ou de outros objetos timedelta com os operadores + e -. Um objeto timedelta pode ser multiplicado ou dividido por valores inteiros ou de ponto flutuante com os operadores * e /. Digite o seguinte no shell interativo:

```
u >>> oct21st = datetime.datetime(2015, 10, 21, 16, 29, 0)
v >>> aboutThirtyYears = datetime.timedelta(days=365 * 30)
  >>> oct21st
  datetime.datetime(2015, 10, 21, 16, 29)
  >>> oct21st - aboutThirtyYears
  datetime.datetime(1985, 10, 28, 16, 29)
  >>> oct21st - (2 * aboutThirtyYears)
  datetime.datetime(1955, 11, 5, 16, 29)
```

Nesse caso, criamos um objeto datetime para o dia 21 de outubro de 2015 u e um objeto timedelta para uma duração de aproximadamente 30 anos (estamos supondo que há 365 dias em cada um desses anos) v. Subtrair aboutThirtyYears de oct21st nos dá um objeto datetime para a data correspondente a 30 anos antes de 21 de outubro de 2015. Subtrair 2 * aboutThirtyYears de oct21st retorna um objeto datetime para a data correspondente a 60 anos antes de 21 de outubro de 2015.

Fazendo uma pausa até uma data específica

O método time.sleep() permite fazer uma pausa em um programa durante um determinado número de segundos. Ao usar um loop while, podemos fazer uma pausa nos programas até uma data específica. Por exemplo, o código a seguir permanecerá no loop até o Halloween (Dia das Bruxas) de 2016:

```
import datetime
import time
halloween2016 = datetime.datetime(2016, 10, 31, 0, 0, 0)
while datetime.datetime.now() < halloween2016:
    time.sleep(1)
```

A chamada a time.sleep(1) provocará uma pausa em seu programa Python para que o computador não desperdice ciclos de processamento de CPU apenas para verificar o horário repetidamente. Em vez disso, o loop while

verificará a condição apenas uma vez por segundo e continuará com o restante do programa após o Halloween de 2016 (ou em qualquer instante que você programá-lo para parar).

Convertendo objetos datetime em strings

Os epoch timestamps e os objetos datetime não são muito amigáveis aos olhares humanos. Utilize o método strftime() para exibir um objeto datetime como uma string. [O *f* no nome da função strftime() quer dizer *format* (formatar).]

O método strftime() utiliza diretivas semelhantes àquelas usadas na formatação de strings em Python. A tabela 15.1 apresenta uma lista completa das diretivas de strftime().

Tabela 15.1 – Diretivas de strftime()

Diretiva de strftime	Significado

%Y	Ano com o século, como em '2014'

%y	Ano sem o século, de '00' a '99' (de 1970 a 2069)

%m	Mês como um número decimal, de '01' a '12'

%B	Nome completo do mês, como em 'November'

%b	Nome abreviado do mês, como em 'Nov'

%d	Dia do mês, de '01' a '31'

%j	Dia do ano, de '001' a '366'

%w	Dia da semana, de '0' (domingo) a '6' (sábado)

%A	Nome completo do dia da semana, como em 'Monday'

%a	Nome do dia da semana abreviado, como em 'Mon'

%H	Hora (relógio com 24 horas), de '00' a '23'

%I	Hora (relógio com 12 horas), de '01' a '12'

%M	Minuto, de '00' a '59'

%S	Segundo, de '00' a '59'

%p	'AM' ou 'PM'

%%	Caractere literal '%'

Passe uma string com formato personalizado a strrftime() contendo diretivas de formatação (juntamente com qualquer barra, dois-pontos e outros caracteres desejados), e essa função retornará as informações do objeto datetime como uma string formatada. Digite o seguinte no shell interativo:

```
>>> oct21st = datetime.datetime(2015, 10, 21, 16, 29, 0)
>>> oct21st.strftime('%Y/%m/%d %H:%M:%S')
'2015/10/21 16:29:00'
>>> oct21st.strftime('%I:%M %p')
'04:29 PM'
>>> oct21st.strftime("%B of '%y")
"October of '15"
```

Nesse caso, temos um objeto datetime para 21 de outubro de 2015 às 16h29min armazenado em oct21st. Se passarmos a string com formato personalizado '%Y/%m/%d %H:%M:%S' a strftime(), uma string contendo 2015, 10 e 21 separados por barras e 16, 29 e 00 separados por dois-pontos será retornada. Se passarmos '%I:%M% p', '04:29 PM' será retornado, e se passarmos "%B of '%y", "October of '15" será retornado. Observe que strftime() não começa com datetime.datetime.

Convertendo strings em objetos datetime

Se você tiver uma string contendo informações de datas, por exemplo, '2015/10/21 16:29:00' ou 'October 21, 2015', e precisar convertê-la em um objeto datetime, utilize a função datetime.datetime.strptime(). A função strptime() faz o inverso do método strftime(). Uma string com formato personalizado que utilize as mesmas diretivas de strftime() deverá ser passada para que strptime() saiba como fazer parse e compreender a string. (O *p* no nome da função strptime() quer dizer *parse*).

Digite o seguinte no shell interativo:

```
u >>> datetime.datetime.strptime('October 21, 2015', '%B %d, %Y')
  datetime.datetime(2015, 10, 21, 0, 0)
  >>> datetime.datetime.strptime('2015/10/21 16:29:00', '%Y/%m/%d
  %H:%M:%S')
  datetime.datetime(2015, 10, 21, 16, 29)
```

```
>>> datetime.datetime.strptime("October of '15", "%B of '%y")
datetime.datetime(2015, 10, 1, 0, 0)
>>> datetime.datetime.strptime("November of '63", "%B of '%y")
datetime.datetime(2063, 11, 1, 0, 0)
```

Para obter um objeto datetime a partir da string 'October 21, 2015', passe essa string como o primeiro argumento de strptime() e a string com formato personalizado que corresponda a 'October 21, 2015' como o segundo argumento u. A string com as informações da data deve corresponder exatamente à string com formato personalizado; do contrário, o Python lançará uma exceção ValueError.

Revisão das funções de tempo do Python

As datas e as horas em Python podem envolver vários tipos diferentes de dados e de funções. A seguir, apresentamos uma revisão dos três tipos diferentes de valores usados para representar tempo:

- Um timestamp Unix epoch (usado pelo módulo time) é um valor de ponto flutuante ou inteiro correspondente ao número de segundos desde a zero hora do dia 1 de janeiro de 1970, UTC.
- Um objeto datetime (do módulo datetime) contém inteiros armazenados nos atributos year, month, day, hour, minute e second.
- Um objeto timedelta (do módulo datetime) representa uma duração, e não um instante específico.

A seguir, apresentamos uma revisão das funções de tempo, seus parâmetros e os valores de retorno:

- A função time.time() retorna um valor de ponto flutuante correspondente ao timestamp epoch do instante atual.
- A função time.sleep(*seconds*) interrompe o programa pela quantidade de segundos especificada pelo argumento *seconds*.
- A função datetime.datetime(*year*, *month*, *day*, *hour*, *minute*, *second*) retorna um objeto datetime para o instante especificado pelos argumentos. Se os argumentos *hour*, *minute* ou *second* não forem especificados, 0 será usado como default.
- A função datetime.datetime.now() retorna um objeto datetime para o instante

atual.

- A função datetime.datetime.fromtimestamp(*epoch*) retorna um objeto datetime do instante representado pelo argumento de timestamp *epoch*.
- A função datetime.timedelta(*weeks*, *days*, *hours*, *minutes*, *seconds*, *milliseconds*, *microseconds*) retorna um objeto timedelta que representa uma duração. Os argumentos nomeados da função são todos opcionais e não incluem *month* ou *year*.
- O método total_seconds() para objetos timedelta retorna o número de segundos representado pelo objeto timedelta.
- O método strftime(*format*) retorna uma string com o tempo representado pelo objeto datetime em um formato personalizado, baseado na string *format*. Consulte a tabela 15.1 para ver os detalhes de formatação.
- A função datetime.datetime.strptime(*time_string*, *format*) retorna um objeto datetime do instante especificado por *time_string*, cujo parse será feito com o argumento de string *format*. Consulte a tabela 15.1 para ver os detalhes de formatação.

Multithreading

Para apresentar o conceito de multithreading (várias threads), vamos dar uma olhada em uma situação de exemplo. Suponha que você queira que um código seja executado após um intervalo de tempo ou em um horário específico. Você poderá acrescentar um código como este no início de seu programa:

```
import time, datetime

startTime = datetime.datetime(2029, 10, 31, 0, 0, 0)
while datetime.datetime.now() < startTime:
   time.sleep(1)

print('Program now starting on Halloween 2029')
--trecho removido--
```

Esse código designa uma data de início igual a 31 de outubro de 2029 e permanece chamando time.sleep(1) até que a data de início seja atingida. Seu

programa não poderá fazer nada enquanto espera o loop com as chamadas a time.sleep() terminar; ele simplesmente ficará esperando até o Halloween de 2029. Isso ocorre porque os programas Python, por padrão, têm uma única *thread* de execução.

Para entender o que é uma thread de execução, lembre-se da discussão do capítulo 2 sobre controle de fluxo, quando você imaginou a execução de um programa como se estivesse colocando seu dedo em uma linha de código de seu programa e o movesse para a próxima linha ou para qualquer local em que a instrução de controle de fluxo o enviasse. Um programa *single-threaded* (com uma única thread) tem apenas um dedo. Porém um programa *multithreaded* (com várias threads) tem vários dedos. Cada dedo continua se movendo para a próxima linha de código, conforme definido pelas instruções de controle de fluxo, porém os dedos podem estar em diferentes lugares do programa, executando linhas de código distintas ao mesmo tempo. (Todos os programas deste livro até agora eram single threaded.)

Em vez de fazer todo o seu código esperar até a função time.sleep() terminar, o código a ser executado após um intervalo de tempo ou agendado para determinado horário poderá estar em uma thread diferente se usarmos o módulo threading do Python. A thread separada fará uma pausa para as chamadas a time.sleep. Enquanto isso, seu programa poderá realizar outras tarefas na thread original.

Para criar uma thread separada, inicialmente você deve criar um objeto Thread ao chamar a função threading.Thread(). Digite o código a seguir e salve o novo arquivo como *threadDemo.py*:

```
  import threading, time
  print('Start of program.')

u def takeANap():
      time.sleep(5)
      print('Wake up!')

v threadObj = threading.Thread(target=takeANap)
w threadObj.start()

  print('End of program.')
```

Em u definimos uma função que queremos usar em uma nova thread. Para criar um objeto Thread, chamamos threading.Thread() e passamos o argumento nomeado target=takeANap v. Isso quer dizer que a função que queremos chamar na nova thread é takeANap(). Observe que o argumento nomeado é target=takeANap, e não target=takeANap(). Isso ocorre porque você deve passar a própria função takeANap() como argumento em vez de chamar takeANap() e passar o seu valor de retorno.

Após armazenarmos o objeto Thread criado por threading.Thread() em threadObj, chamamos threadObj.start() w para criar a nova thread e iniciar a execução da função-alvo na nova thread. Quando esse programa for executado, a saída terá o seguinte aspecto:

```
Start of program.
End of program.
Wake up!
```

Isso pode parecer um pouco confuso. Se print('End of program.') é a última linha do programa, você poderia achar que ela deveria ser a última informação exibida. O motivo pelo qual Wake up! é apresentado depois deve-se ao fato de que, quando threadObj.start() é chamado, a função-alvo de threadObj é executada em uma nova thread de execução. Pense nisso como um segundo dedo que apareça no início da função takeANap(). A thread principal continua em print('End of program.'). Enquanto isso, a nova thread que estava executando a chamada a time.sleep(5) faz uma pausa de cinco segundos. Depois de acordar de seu cochilo de cinco segundos, ela exibirá 'Wake up!' e retornará da função takeANap(). Cronologicamente, 'Wake up!' é a última informação exibida pelo programa.

Normalmente, um programa termina quando a última linha de código do arquivo é executada (ou a função sys.exit() é chamada). Entretanto *threadDemo.py* tem duas threads. A primeira é a thread original, que começa no início do programa e termina após print(‹End of program.›). A segunda thread é criada quando threadObj.start() é chamada, começa no início da função takeANap() e termina após takeANap() retornar.

Um programa Python não terminará até que todas as suas threads tenham terminado. Ao executar *threadDemo.py*, apesar de a thread original ter

terminado, a segunda thread ainda estava executando a chamada a time.sleep(5).

Passando argumentos à função-alvo da thread

Se a função-alvo que você quer executar na nova thread aceitar argumentos, eles poderão ser passados à função threading.Thread(). Por exemplo, suponha que você queira executar a chamada a print() a seguir em sua própria thread:

```
>>> print('Cats', 'Dogs', 'Frogs', sep=' & ')
Cats & Dogs & Frogs
```

Essa chamada a print() tem três argumentos normais, que são 'Cats', 'Dogs' e 'Frogs', e um argumento nomeado sep=' & '. Os argumentos normais podem ser passados como uma lista ao argumento nomeado args em threading.Thread(). O argumento nomeado pode ser especificado como um dicionário ao argumento nomeado kwargs em threading.Thread().

Digite o seguinte no shell interativo:

```
>>> import threading
>>> threadObj = threading.Thread(target=print, args=['Cats', 'Dogs', 'Frogs'],
kwargs={'sep': ' & '})
>>> threadObj.start()
Cats & Dogs & Frogs
```

Para garantir que os argumentos 'Cats', 'Dogs' e 'Frogs' sejam passados a print() na nova thread, passamos args=['Cats', 'Dogs', 'Frogs'] para threading.Thread(). Para garantir que o argumento nomeado sep=' & ' seja passado a print() na nova thread, passamos kwargs={'sep': '& '} para threading.Thread().

A chamada a threadObj.start() criará uma nova thread para chamar a função print(), 'Cats', 'Dogs' e 'Frogs' serão passados como argumentos e ' & ' será passado ao argumento nomeado sep.

Esta é uma maneira incorreta de criar uma nova thread que chame print():

```
threadObj = threading.Thread(target=print('Cats', 'Dogs', 'Frogs', sep=' & '))
```

Essa instrução acaba chamando a função print() e passando seu valor de retorno (o valor de retorno de print() é sempre None) como o argumento nomeado target. Ela *não* passa a função print() propriamente dita. Ao passar

argumentos a uma função em uma nova thread, utilize os argumentos nomeados args e kwargs da função threading.Thread().

Problemas de concorrência

Novas threads podem ser facilmente criadas e executadas ao mesmo tempo. No entanto a existência de várias threads pode causar *problemas de concorrência*. Esses problemas ocorrem quando as threads leem e escrevem em variáveis ao mesmo tempo, fazendo as threads tropeçarem umas nas outras. Os problemas de concorrência podem ser difíceis de reproduzir de forma consistente, tornando-as complicadas de depurar.

A programação multithreaded por si só é um assunto amplo e está além do escopo deste livro. Para evitar problemas de concorrência, jamais permita que várias threads leiam ou escrevam nas mesmas variáveis. Ao criar um novo objeto Thread, certifique-se de que sua função-alvo utilize somente variáveis locais nessa função. Isso evitará programas de concorrência difíceis de depurar em seus programas.

NOTA Um tutorial sobre programação multithreaded para iniciantes está disponível em *http://nostarch.com/automatestuff/*.

Projeto: Programa multithreaded para download de XKCD

No capítulo 11, criamos um programa que fazia o download de todas as tirinhas do site XKCD. Esse programa era single-threaded: ele baixava uma tirinha de cada vez. A maior parte do tempo de execução do programa era consumida para estabelecer a conexão com a rede, iniciar o download e gravar as imagens baixadas no disco rígido. Se você tiver uma conexão de banda larga com a Internet, seu programa single-threaded não estará utilizando toda a largura de banda disponível.

Um programa multithreaded que tenha algumas threads baixando tirinhas enquanto outras estão estabelecendo conexões e gravando os arquivos com as imagens das tirinhas em disco usará sua conexão de Internet de maneira muito mais eficiente e baixará a coleção de tirinhas mais rapidamente. Abra uma nova janela no editor de arquivo e salve esse arquivo como

multidownloadXkcd.py. Você modificará esse programa para adicionar multithreading. O código-fonte com as alterações completas está disponível para download em *http://nostarch.com/automatestuff/*.

Passo 1: Modificar o programa para que use uma função

Esse programa, em sua maior parte, conterá o mesmo código de downloading do capítulo 11, portanto vou pular a explicação para os códigos com Requests e BeautifulSoup. As principais alterações necessárias são importar o módulo threading e criar uma função downloadXkcd() que aceite os números das tirinhas inicial e final como parâmetros.

Por exemplo, chamar downloadXkcd(140, 280) executará um loop com o código de download para baixar as tirinhas em *http://xkcd.com/140*, *http://xkcd.com/141*, *http://xkcd.com/142* e assim por diante, até *http://xkcd.com/279*. Cada thread criada chamará downloadXkcd() e passará um intervalo diferente de tirinhas para ser baixado.

Acrescente o código a seguir em seu programa *multidownloadXkcd.py*:

```
   #! python3
   # multidownloadXkcd.py – Faz download das tirinhas XKCD usando várias threads.

   import requests, os, bs4, threading
u os.makedirs('xkcd', exist_ok=True)    # armazena as tirinhas em ./xkcd

v def downloadXkcd(startComic, endComic):
w     for urlNumber in range(startComic, endComic):
          # Faz download da página.
          print('Downloading page http://xkcd.com/%s...' % (urlNumber))
x         res = requests.get('http://xkcd.com/%s' % (urlNumber))
          res.raise_for_status()

y         soup = bs4.BeautifulSoup(res.text)

          # Encontra o URL da imagem da tirinha.
z         comicElem = soup.select('#comic img')
          if comicElem == []:
              print('Could not find comic image.')
          else:
```

```
{           comicUrl = comicElem[0].get('src')
            # Faz o download da imagem.
            print('Downloading image %s...' % (comicUrl))
|           res = requests.get(comicUrl)
            res.raise_for_status()

            # Salva a imagem em ./xkcd.
            imageFile = open(os.path.join('xkcd', os.path.basename(comicUrl)), 'wb')
            for chunk in res.iter_content(100000):
                imageFile.write(chunk)
            imageFile.close()

# TODO: Cria e inicia os objetos Thread.
# TODO: Espera todas as threads terminarem.
```

Após importar os módulos necessários, criamos um diretório para armazenar as tirinhas em u e começamos a definir downloadxkcd() v. Percorremos todos os números no intervalo especificado em um loop w e fazemos download de cada página x. Utilizamos o Beautiful Soup para analisar o HTML de cada página y e encontrar a imagem da tirinha z. Se nenhuma imagem de tirinha for encontrada em uma página, exibiremos uma mensagem. Caso contrário, o URL da imagem é obtido { e o download da imagem é feito |. Por fim, salvamos a imagem no diretório que criamos.

Passo 2: Criar e iniciar as threads

Agora que definimos downloadXkcd(), criaremos as diversas threads que chamarão essa função para fazer download de intervalos diferentes de tirinhas do site XKCD. Acrescente o código a seguir em *multidownloadXkcd.py* após a definição da função downloadXkcd():

```
#! python3
# multidownloadXkcd.py – Faz download das tirinhas XKCD usando várias threads.

--trecho removido--

# Cria e inicia os objetos Thread.
downloadThreads = []            # uma lista com todos os objetos Thread
for i in range(0, 1400, 100):    # executa o loop 14 vezes e cria 14 threads
    downloadThread = threading.Thread(target=downloadXkcd, args=(i, i + 99))
```

```
    downloadThreads.append(downloadThread)
    downloadThread.start()
```

Inicialmente, criamos uma lista downloadThreads vazia; a lista nos ajudará a manter o controle dos diversos objetos Thread que criaremos. Em seguida, iniciamos o nosso loop for. A cada passagem pelo loop, criamos um objeto Thread com threading.Thread(), adicionamos o objeto Thread à lista e chamamos start() para começar a executar downloadXkcd() na nova thread. Como o loop for define a variável i de 0 a 1400 em passos de 100, i será definido com 0 na primeira iteração, com 100 na segunda iteração, com 200 na terceira, e assim sucessivamente. Pelo fato de passarmos args=(i, i + 99) a threading.Thread(), os dois argumentos passados para downloadXkcd() serão 0 e 99 na primeira iteração, 100 e 199 na segunda iteração, 200 e 299 na terceira, e assim por diante.

À medida que o método start() do objeto Thread for chamado e a nova thread começar a executar o código em downloadXkcd(), a thread principal prosseguirá para a próxima iteração do loop for e a thread seguinte será criada.

Passo 3: Esperar todas as threads terminarem

A thread principal prosseguirá normalmente enquanto as demais threads que criamos fazem download das tirinhas. No entanto suponha que há um código que você não queira executar na thread principal até que todas as threads tenham sido concluídas. Chamar o método join() de um objeto Thread deixará o código bloqueado até essa thread ter finalizado. Ao usar um loop for para fazer uma iteração por todos os objetos Thread na lista downloadThreads, a thread principal poderá chamar o método join() em cada uma das demais threads. Acrescente o código a seguir no final de seu programa:

```
#! python3
# multidownloadXkcd.py – Faz download das tirinhas XKCD usando várias threads.

--trecho removido--

# Espera todas as threads terminarem.
for downloadThread in downloadThreads:
```

```
    downloadThread.join()
print('Done.')
```

A string 'Done.' não será exibida até que todas as chamadas a join() tenham retornado. Se um objeto Thread já tiver terminado quando seu método join() for chamado, o método simplesmente retornará de imediato. Se quiser ampliar esse programa com um código que execute somente depois que todas as tirinhas tiverem sido baixadas, substitua a linha print('Done.') pelo seu novo código.

Iniciando outros programas a partir do Python

Seu programa Python pode iniciar outros programas em seu computador com a função Popen() do módulo interno subprocess. (O *P* no nome da função Popen() quer dizer *process*, ou seja, processo.) Se você tiver várias instâncias abertas de uma aplicação, cada uma dessas instâncias será um processo separado do mesmo programa. Por exemplo, se você abrir várias janelas de seu navegador web ao mesmo tempo, cada uma dessas janelas será um processo diferente do programa de navegador web. Veja a figura 15.1, que mostra um exemplo de vários processos de calculadora abertos de uma só vez.

Cada processo pode ter várias threads. De modo diferente das threads, um processo não pode ler nem escrever diretamente nas variáveis de outro processo. Se você pensar em um programa multithreaded como tendo diversos dedos seguindo o código-fonte, então ter diversos processos do mesmo programa abertos é como ter uma amiga com uma cópia separada do código-fonte do programa. Ambos estarão executando independentemente o mesmo programa.

Se quiser iniciar um programa externo a partir de seu script Python, passe o nome do arquivo do programa para subprocess.Popen(). (No Windows, clique com o botão direito do mouse no item de menu Start (Iniciar) da aplicação e selecione Properties (Propriedades) para ver o nome do arquivo da aplicação. No OS X, dê um CTRL-clique na aplicação e selecione Show Package Contents (Mostrar Conteúdo do Pacote) para encontrar o path do arquivo executável.) A função Popen() retornará imediatamente. Tenha em mente que

o programa iniciado não será executado na mesma thread que o seu programa Python.

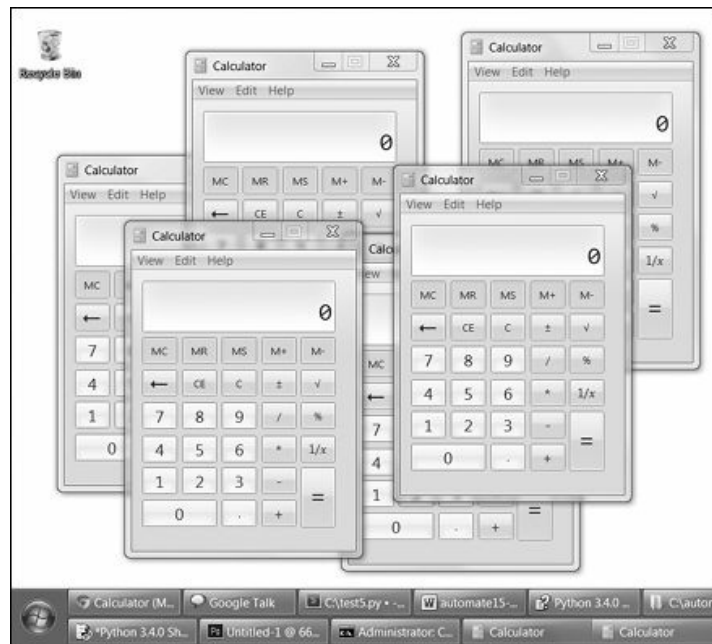



Figura 15.1 – Seis processos do mesmo programa de calculadora em execução.

Em um computador Windows, digite o seguinte no shell interativo:

```
>>> import subprocess
>>> subprocess.Popen('C:\\Windows\\System32\\calc.exe')
<subprocess.Popen object at 0x0000000003055A58>
```

No Ubuntu Linux, digite o seguinte:

```
>>> import subprocess
>>> subprocess.Popen('/usr/bin/gnome-calculator')
<subprocess.Popen object at 0x7f2bcf93b20>
```

No OS X, o processo será um pouco diferente. Veja a seção “Abrindo arquivos com aplicativos default”.

O valor de retorno é um objeto Popen que tem dois métodos úteis: poll() e wait().

Podemos pensar no método poll() como se você estivesse perguntando a uma amiga se ela acabou de executar o código que você lhe deu. O método poll() retornará None se o processo ainda estiver executando quando poll() for chamado. Se o programa tiver terminado, um inteiro referente ao *código de*

saída (exit code) do processo será retornado. Um código de saída é usado para indicar se o processo terminou sem erros (um código de saída igual a 0) ou se um erro fez o processo terminar (um código de saída diferente de zero – geralmente é 1, mas pode variar conforme o programa).

O método wait() é como esperar sua amiga acabar de usar seu código antes que você possa trabalhar no seu. O método wait() ficará bloqueado até que o processo iniciado tenha terminado. Isso é útil se você quiser que seu programa faça uma pausa até que o usuário acabe de usar o outro programa. O valor de retorno de wait() é um inteiro com o código de saída do processo.

No Windows, digite as instruções a seguir no shell interativo. Observe que a chamada a wait() ficará bloqueada até que você saia do programa de calculadora iniciado.

```
u >>> calcProc = subprocess.Popen('c:\\Windows\\System32\\calc.exe')
v >>> calcProc.poll() == None
  True
w >>> calcProc.wait()
  0
  >>> calcProc.poll()
  0
```

Nesse caso, iniciamos um processo para a calculadora u. Enquanto ele estiver executando, verificamos se poll() retorna None v. Esse valor deverá ser retornado, pois o processo ainda está executando. Em seguida, fechamos o programa de calculadora e chamamos wait() no processo terminado w. wait() e poll() agora retornam 0, indicando que o processo terminou sem erros.

Passando argumentos da linha de comando a Popen()

Podemos passar argumentos de linha de comando aos processos criados com Popen(). Para isso, passe uma lista como único argumento de Popen(). A primeira string dessa lista será o nome do arquivo executável do programa que você quer iniciar; todas as strings subsequentes serão os argumentos de linha de comando a serem passados para o programa quando ele iniciar. Essa lista será o valor de sys.argv para o programa iniciado.

A maioria das aplicações com GUI (Graphical User Interface, ou Interface gráfica de usuário) não utiliza argumentos de linha de comando com tanta

frequência quanto os programas baseados em linha de comando ou em terminal. No entanto a maioria das aplicações com GUI aceitará um único argumento com um arquivo que será imediatamente aberto pelas aplicações quando elas iniciarem. Por exemplo, se você estiver usando Windows, crie um arquivo-texto simples chamado *C:\hello.txt* e digite o seguinte no shell interativo:

```
>>> subprocess.Popen(['C:\\Windows\\notepad.exe', 'C:\\hello.txt'])
<subprocess.Popen object at 0x00000000032DCEB8>
```

Isso não só fará o aplicativo Notepad ser iniciado como também o fará abrir o arquivo *C:\hello.txt* imediatamente.

Task Scheduler, launchd e cron

Se tiver bastante experiência com computadores, talvez você conheça o Task Scheduler (Agendador de Tarefas) do Windows, o launchd do OS X ou a ferramenta de agendamento cron do Linux. Essas ferramentas bem documentadas e confiáveis permitem agendar aplicações para que sejam iniciadas em instantes específicos. Se quiser saber mais sobre elas, você poderá encontrar links para tutoriais em *http://nostarch.com/automatestuff/*.

Usar a ferramenta de agendamento pronta de seu sistema operacional evitará que seja preciso escrever seu próprio código de verificação de relógio para agendar seus programas. No entanto utilize a função time.sleep() se tiver de fazer somente uma pausa breve em seu programa. Em vez de usar a ferramenta de agendamento do sistema operacional, seu código poderá executar um loop até determinada data e hora, chamando time.sleep(1) sempre que passar pelo loop.

Abrindo sites com o Python

Em vez de abrir a aplicação de navegador com subprocess.Popen(), a função webbrowser.open() pode ser usada para iniciar um navegador web em um site específico a partir de seu programa. Veja a seção “Projeto: *mapIt.py* com o módulo webbrowser” para obter mais detalhes.

Executando outros scripts Python

Você pode iniciar um script Python a partir do Python, como qualquer outra aplicação. Basta passar o executável *python.exe* para Popen() e o nome do arquivo contendo o script *.py* que você deseja executar como argumento. Por exemplo, a instrução a seguir executará o script *hello.py* do capítulo 1:

```
>>> subprocess.Popen(['C:\\python34\\python.exe', 'hello.py'])
<subprocess.Popen object at 0x000000000331CF28>
```

Passe uma lista contendo uma string com o path do executável Python e uma string com o nome do arquivo de script a Popen(). Se o script que você estiver iniciando precisar de argumentos de linha de comando, adicione-os à lista após o nome do arquivo de script. A localização do executável do Python no Windows é *C:\python34\python.exe*. No OS X é */Library/Frameworks/Python.framework/Versions/3.3/bin/python3* e no Linux é */usr/bin/python3*.

De modo diferente de importar o programa Python como um módulo, quando seu programa Python iniciar outro programa Python, ambos serão executados em processos separados e não poderão compartilhar variáveis um com o outro.

Abrindo arquivos com aplicativos default

Dar um clique duplo em um arquivo *.txt* em seu computador fará o aplicativo associado à extensão de arquivo *.txt* ser iniciado automaticamente. Seu computador terá várias dessas associações com extensões de arquivo já definidas. O Python também pode abrir arquivos dessa maneira com o Popen().

Cada sistema operacional tem um programa que executa o equivalente a dar um clique duplo em um arquivo de documento para abri-lo. No Windows, esse programa é o start. No OS X, é open, e no Ubuntu Linux, é see. Digite o seguinte no shell interativo, passando 'start', 'open' ou 'see' a Popen(), de acordo com o seu sistema:

```
>>> fileObj = open('hello.txt', 'w')
>>> fileObj.write('Hello world!')
12
>>> fileObj.close()
```

```
>>> import subprocess
>>> subprocess.Popen(['start', 'hello.txt'], shell=True)
```

A FILOSOFIA UNIX

Os programas com um bom design para serem iniciados por outros programas se tornam mais eficazes que seus códigos sozinhos. A *filosofia Unix* é um conjunto de princípios de design de software definido pelos programadores do sistema operacional Unix (com base no qual o Linux e o OS X modernos foram desenvolvidos). Essa filosofia define que é melhor criar programas pequenos, com propósitos limitados e que possam proporcionar interoperabilidade, em vez de criar aplicações grandes, cheias de recursos. Programas menores são mais fáceis de entender e, por permitirem interoperabilidade, eles poderão constituir blocos de construção para aplicações que sejam muito mais eficazes.

Os aplicativos de smartphones também seguem essa abordagem. Se seu aplicativo de restaurantes precisar exibir informações sobre como chegar a uma lanchonete, os desenvolvedores não deverão reinventar a roda criando seu próprio código para mapas. O aplicativo de restaurantes simplesmente iniciará um aplicativo de mapas passando-lhe o endereço da lanchonete, assim como seu código Python chamaria uma função e lhe passaria seus argumentos.

A maioria dos programas Python que criamos neste livro se enquadra na filosofia Unix, em especial quanto a um aspecto importante: eles usam argumentos de linha de comando em vez de fazer chamadas à função input(). Se todas as informações necessárias ao seu programa puderem ser fornecidas previamente, será preferível que essas informações sejam passadas como argumentos de linha de comando em vez de esperar o usuário digitá-las. Dessa maneira, os argumentos de linha de comando poderão ser fornecidos por uma pessoa ou por outro programa. Essa abordagem que proporciona interoperabilidade tornará seus programas reutilizáveis como parte de outro programa.

A única exceção é que você não deve passar senhas como argumentos de linha de comando, pois a linha de comando pode registrá-las como parte de seu recurso de histórico de comandos. Em vez disso, seu programa deverá chamar a função input() quando precisar que uma senha seja fornecida.

Você pode ler mais sobre a filosofia Unix em *https://en.wikipedia.org/wiki/Unix_philosophy/*.

Nesse caso, escrevemos Hello world! em um novo arquivo *hello.txt*. Em seguida, chamamos Popen(), passando-lhe uma lista contendo o nome do programa (nesse exemplo, 'start' para Windows) e o nome do arquivo. Também passamos o argumento nomeado shell=True, necessário somente no

Windows. O sistema operacional conhece todas as associações com arquivos e pode determinar que deverá, por exemplo, iniciar o *Notepad.exe* para tratar o arquivo *hello.txt*.

No OS X, o programa open é usado tanto para abrir arquivos de documentos quanto para programas. Digite o seguinte no shell interativo se você tiver um Mac:

```
>>> subprocess.Popen(['open', '/Applications/Calculator.app/'])
<subprocess.Popen object at 0x10202ff98>
```

O aplicativo Calculator (Calculadora) deverá ser aberto.

Projeto: Programa simples de contagem regressiva

Assim como é difícil encontrar um aplicativo simples de cronômetro, pode ser difícil encontrar um aplicativo simples de contagem regressiva. Vamos criar um programa de contagem regressiva que toque um alarme no final da contagem.

De modo geral, o seu programa deverá:

- Fazer uma contagem regressiva a partir de 60.
- Reproduzir um arquivo de áudio (*alarm.wav*) quando a contagem regressiva atingir zero.

Isso significa que seu código deverá:

- Fazer uma pausa de um segundo entre a exibição de cada número na contagem regressiva ao chamar time.sleep().
- Chamar subprocess.Popen() para abrir o arquivo de áudio com o aplicativo default.

Abra uma nova janela no editor de arquivo e salve esse arquivo como *countdown.py*.

Passo 1: Fazer a contagem regressiva

Esse programa exigirá o módulo time para a função time.sleep() e o módulo subprocess para a função subprocess.Popen(). Digite o código a seguir e salve o arquivo como *countdown.py*:

```
#! python3
# countdown.py – Um script simples para contagem regressiva.

import time, subprocess

u timeLeft = 60
  while timeLeft > 0:
v     print(timeLeft, end='')
w     time.sleep(1)
x     timeLeft = timeLeft - 1

# TODO: No final da contagem regressiva, reproduz um arquivo de áudio.
```

Após importar time e subprocess, crie uma variável chamada timeLeft para armazenar o número de segundos restantes na contagem regressiva u. A contagem pode iniciar em 60 – ou você poderá alterar esse número para qualquer valor necessário ou até mesmo obtê-lo a partir de um argumento da linha de comando.

Em um loop while, exiba o tempo remanescente v, faça uma pausa de um segundo w e decremente a variável timeLeft x antes que o loop inicie novamente. O loop continuará executando enquanto timeLeft for maior que 0. Depois disso, a contagem regressiva estará terminada.

Passo 2: Reproduzir o arquivo de áudio

Embora haja módulos de terceiros para reproduzir arquivos de áudio de diversos formatos, a maneira rápida e fácil consiste em simplesmente iniciar qualquer aplicativo que o usuário já utilize para reproduzir arquivos de áudio. O sistema operacional descobrirá, a partir da extensão de arquivo *.wav*, qual aplicativo deverá ser iniciado para reproduzir o arquivo. Em vez de ser um *.wav*, esse arquivo poderá ter outro formato como *.mp3* ou *.ogg*.

Você pode usar qualquer arquivo de áudio que esteja em seu computador para ser reproduzido no final da contagem regressiva ou poderá fazer download de *alarm.wav* a partir de *http://nostarch.com/automatestuff/*.

Adicione o código a seguir:

```
#! python3
# countdown.py – Um script simples para contagem regressiva.
```

```
import time, subprocess

--trecho removido--

# No final da contagem regressiva, reproduz um arquivo de áudio.
subprocess.Popen(['start', 'alarm.wav'], shell=True)
```

Depois que o loop while terminar, *alarm.wav* (ou o arquivo de áudio escolhido por você) será reproduzido para notificar o usuário de que a contagem regressiva acabou. No Windows, não se esqueça de incluir 'start' à lista passada para Popen() e passar o argumento nomeado shell=True. No OS X, passe 'open' no lugar de 'start' e remova shell=True.

Em vez de reproduzir um arquivo de áudio, você poderá salvar um arquivo-texto em algum lugar com uma mensagem como *Break time is over!* (Hora do intervalo acabou!) e usar Popen() para abri-lo no final da contagem regressiva. Isso criará uma janela pop-up com uma mensagem. Você também poderá usar a função webbrowser.open() para abrir um site específico no final da contagem regressiva. De modo diferente dos aplicativos gratuitos de contagem regressiva encontrados online, você poderá usar o que quiser como alarme em seu próprio programa de contagem regressiva!

Ideias para programas semelhantes

Uma contagem regressiva é somente uma espera antes de prosseguir com a execução do programa. Isso também pode ser usado em outras aplicações e outros recursos, por exemplo:

- Usar time.sleep() para dar uma oportunidade ao usuário de pressionar CTRL-C para cancelar uma ação, por exemplo, apagar arquivos. Seu programa poderá exibir uma mensagem “Tecle CTRL-C para cancelar” e, em seguida, tratar qualquer exceção KeyboardInterrupt com instruções try e except.
- Para uma contagem regressiva de longa duração, objetos timedelta podem ser usados para determinar o número de dias, horas, minutos e segundos até um instante específico (um aniversário ou a data de seu aniversário de casamento) no futuro.

Resumo

O Unix epoch (zero hora de 1 de janeiro de 1970, UTC) é uma referência-padrão de tempo para muitas linguagens de programação, incluindo o Python. Embora a função `time.time()` retorne um timestamp epoch (ou seja, um valor de ponto flutuante referente ao número de segundos desde o Unix epoch), o módulo `datetime` é melhor para realizar operações aritméticas com datas e formatar ou fazer parse de strings com informações de datas.

A função `time.sleep()` ficará bloqueada (ou seja, não retornará) durante determinado número de segundos. Ela pode ser usada para adicionar pausas em seu programa. Porém, se quiser agendar seus programas para que iniciem em determinado instante, as instruções em *http://nostarch.com/automatestuff/* poderão informar de que modo a ferramenta de agendamento disponibilizada pelo seu sistema operacional poderá ser usada.

O módulo `threading` é usado para criar várias threads, o que é útil quando é necessário fazer download de diversos arquivos ou realizar outras tarefas simultaneamente. Porém certifique-se de que as threads leiam e escrevam somente em variáveis locais; do contrário, você poderá se deparar com problemas de concorrência.

Por fim, seus programas Python podem iniciar outras aplicações com a função `subprocess.Popen()`. Argumentos de linha de comando podem ser passados para a chamada a `Popen()` para abrir documentos específicos com a aplicação. De modo alternativo, os programas `start`, `open` ou `see` podem ser usados com `Popen()` para que as associações de arquivos feitas pelo seu computador sejam usadas a fim de descobrir automaticamente qual aplicativo deverá ser utilizado para abrir um documento. Ao usar outros aplicativos de seu computador, seus programas Python poderão tirar proveito das capacidades desses aplicativos para atender às suas necessidades de automação.

Exercícios práticos

1. O que é o Unix epoch (Era Unix ou Época Unix)?
2. Qual função retorna o número de segundos desde o Unix epoch?

3. Como podemos fazer uma pausa de exatamente cinco segundos em um programa?
4. O que a função round() retorna?
5. Qual é a diferença entre um objeto datetime e um objeto timedelta?
6. Suponha que você tenha uma função chamada spam(). Como podemos chamar essa função e executar seu código em uma thread separada?
7. O que devemos fazer para evitar problemas de concorrência com várias threads?
8. Como podemos fazer seu programa Python executar o programa *calc.exe* localizado na pasta *C:\Windows\System32* ?

Projetos práticos

Para exercitar, escreva programas que façam as tarefas a seguir.

Cronômetro elegante

Amplie seu projeto de cronômetro deste capítulo para que os métodos de string rjust() e ljust() sejam usados e deixem a saída mais elegante. (Esses métodos foram discutidos no capítulo 6.) Em vez de uma saída como esta:

```
Lap #1: 3.56 (3.56)
Lap #2: 8.63 (5.07)
Lap #3: 17.68 (9.05)
Lap #4: 19.11 (1.43)
```

ela deverá ser semelhante a:

```
Lap # 1:   3.56 (  3.56)
Lap # 2:   8.63 (  5.07)
Lap # 3:  17.68 (  9.05)
Lap # 4:  19.11 (  1.43)
```

Observe que você precisará de versões em string das variáveis inteiras e de ponto flutuante lapNum, lapTime e totalTime para chamar os métodos de string nessas variáveis.

Em seguida, utilize o módulo pyperclip apresentado no capítulo 6 para copiar o texto de saída para o clipboard (área de transferência) de modo que o

usuário possa colar a saída rapidamente em um arquivo-texto ou em um email.

Programa agendado para fazer download de web comics

Crie um programa que verifique os sites de diversos web comics e faça downloads automaticamente das imagens se a tirinha tiver sido atualizada desde o último acesso do programa. A ferramenta de agendamento de seu sistema operacional (Scheduled Tasks no Windows, launchd no OS X e cron no Linux) poderá executar seu programa Python uma vez ao dia. O programa Python propriamente dito poderá fazer download da tirinha e copiá-la para o seu desktop para que seja fácil encontrá-la. Isso evitará que você precise verificar o site por conta própria para ver se ele foi atualizado. (Uma lista de web comics está disponível em *http://nostarch.com/automatestuff/*.)

CAPÍTULO 16

ENVIANDO EMAIL E MENSAGENS DE TEXTO

Verificar emails e respondê-los é uma tarefa que consome bastante tempo. É claro que você não pode simplesmente criar um programa que cuide de todos os seus emails, pois cada mensagem exige sua própria resposta. Entretanto muitas das tarefas relacionadas a emails poderão ser automatizadas se você souber como criar programas que possam enviar e receber emails.

Por exemplo, talvez você tenha uma planilha cheia de registros de clientes e queira enviar uma carta formal a cada um deles de acordo com suas idades e os detalhes de sua localização. Os softwares comerciais podem não ser capazes de fazer isso para você; felizmente, é possível criar seu próprio programa para enviar esses emails, economizando bastante tempo que seria consumido para copiar e colar ao gerar os emails.

Você também pode criar programas para enviar emails e textos de SMS para você mesmo e receber uma notificação mesmo quando estiver distante de seu computador. Se estiver automatizando uma tarefa que demore algumas horas para ser realizada, você não vai querer retornar ao seu computador a intervalos de alguns minutos para conferir o status do programa. Em vez disso, seu programa poderá simplesmente enviar uma mensagem de texto ao seu telefone quando tiver encerrado – deixando você livre para focar tarefas mais importantes enquanto estiver distante de seu computador.

SMTP

Assim como o HTTP é o protocolo usado pelos computadores para enviar páginas web pela Internet, o SMTP (*Simple Mail Transfer Protocol*) é o protocolo usado para enviar emails. O SMTP define o modo como as mensagens de email devem ser formatadas, criptografadas e trocadas entre servidores de emails, além de todos os demais detalhes com os quais seu computador deve lidar após você clicar em Send (Enviar). Contudo não é

preciso conhecer esses detalhes técnicos, pois o módulo smtplib do Python os simplifica por meio de algumas funções.

O SMTP simplesmente cuida de enviar emails a outras pessoas. Um protocolo diferente chamado IMAP cuida da obtenção dos emails enviados a você e será descrito na seção "IMAP".

Enviando emails

Talvez você já tenha familiaridade com o envio de emails a partir do Outlook ou do Thunderbird ou por meio de um site como o Gmail ou o Yahoo! Mail. Infelizmente, o Python não oferece uma interface gráfica de usuário elegante como esses serviços. Em vez disso, você deve chamar funções para executar cada passo principal do SMTP, conforme mostrado no exemplo a seguir do shell interativo.

> **NOTA** Não execute esse exemplo no IDLE; ele não funcionará, pois *smtp.example.com*, *bob@example.com*, *MY_SECRET_PASSWORD* e *alice@example.com* são apenas placeholders. Esse código apresenta somente uma visão geral do processo de envio de emails com o Python.

```
>>> import smtplib
>>> smtpObj = smtplib.SMTP('smtp.example.com', 587)
>>> smtpObj.ehlo()
(250, b'mx.example.com at your service, [216.172.148.131]\nSIZE
35882577\n8BITMIME\nSTARTTLS\nENHANCEDSTATUSCODES\nCHUNKING')
>>> smtpObj.starttls()
(220, b'2.0.0 Ready to start TLS')
>>> smtpObj.login('bob@example.com', 'MY_SECRET_PASSWORD')
(235, b'2.7.0 Accepted')
>>> smtpObj.sendmail('bob@example.com', 'alice@example.com', 'Subject: So
long.\nDear Alice, so long and thanks for all the fish. Sincerely, Bob')
{}
>>> smtpObj.quit()
(221, b'2.0.0 closing connection ko10sm23097611pbd.52 - gsmtp')
```

Nas seções seguintes, descreveremos cada passo, substituindo os placeholders pelas suas informações para conectar-se a um servidor SMTP e fazer login, enviar um email e desconectar-se do servidor.

Conectando-se a um servidor SMTP

Se você já configurou o Thunderbird, o Outlook ou outro programa para se conectar à sua conta de email, talvez tenha familiaridade com a configuração do servidor SMTP e da porta. Essas configurações serão diferentes para cada provedor de emails, porém uma pesquisa na web em busca de *<seu provedor> smtp configurações* deverá resultar no servidor e na porta a serem usados.

O nome de domínio do servidor SMTP normalmente será o nome de domínio de seu provedor de emails com *smtp.* na frente. Por exemplo, o servidor SMTP do Gmail está em *smtp.gmail.com*. A tabela 16.1 lista alguns provedores comuns de emails e seus servidores SMTP. (A porta é um valor inteiro e quase sempre será 587, que é usado pelo padrão de criptografia de comandos – o TLS.)

Tabela 16.1 – Provedores de email e seus servidores SMTP

Provedor	Nome de domínio do servidor SMTP
Gmail	*smtp.gmail.com*
Outlook.com/Hotmail.com	*smtp-mail.outlook.com*
Yahoo Mail	*smtp.mail.yahoo.com*
AT&T	*smpt.mail.att.net* (porta 465)
Comcast	*smtp.comcast.net*
Verizon	*smtp.verizon.net* (porta 465)

De posse do nome de domínio e da informação da porta para o seu provedor de emails, crie um objeto `SMTP` ao chamar `smptlib.SMTP()`, passando-lhe o nome do domínio como um argumento do tipo string e a porta como um argumento inteiro. O objeto `SMTP` representa uma conexão com um servidor de emails SMTP e contém métodos para enviar emails. Por exemplo, a chamada a seguir cria um objeto `SMTP` para fazer uma conexão com o Gmail:

```
>>> smtpObj = smtplib.SMTP('smtp.gmail.com', 587)
>>> type(smtpObj)
<class 'smtplib.SMTP'>
```

Executar `type(smtpObj)` mostra que há um objeto SMTP armazenado em `smtpObj`. Você precisará desse objeto `SMTP` para chamar os métodos para fazer

login e enviar emails. Se a chamada a smptlib.SMTP() não for bem-sucedida, é sinal de que seu servidor SMTP talvez não suporte TLS na porta 587. Nesse caso, será necessário criar um objeto SMTP usando smtplib.SMTP_SSL() e a porta 465.

```
>>> smtpObj = smtplib.SMTP_SSL('smtp.gmail.com', 465)
```

NOTA Se você não estiver conectado à Internet, o Python lançará uma exceção socket.gaierror: [Errno 11004] getaddrinfo failed ou algo semelhante.

Em seus programas, as diferenças entre TLS e SSL não serão importantes. Basta saber qual é o padrão de criptografia utilizado pelo seu servidor SMTP para saber como se conectar a ele. Em todos os exemplos a seguir no shell interativo, a variável smtpObj conterá um objeto SMTP retornado pela função smtplib.SMTP() ou pela função smtplib.SMTP_SSL().

Enviando a mensagem “Hello” do SMTP

Depois que você tiver o objeto SMTP, chame seu método ehlo(), de nome inusitado, para “dizer alô” ao servidor de emails SMTP. Essa saudação é o primeiro passo no SMTP e é importante para estabelecer uma conexão com o servidor. Não é preciso conhecer as especificidades desses protocolos. Basta garantir que o método ehlo() seja inicialmente chamado após obter o objeto SMTP; do contrário, as chamadas de métodos feitas posteriormente resultarão em erros. A seguir, apresentamos um exemplo de uma chamada a ehlo() e seu valor de retorno:

```
>>> smtpObj.ehlo()
(250, b'mx.google.com at your service, [216.172.148.131]\nSIZE
35882577\n8BITMIME\nSTARTTLS\nENHANCEDSTATUSCODES\nCHUNKING')
```

Se o primeiro item da tupla retornada for o inteiro 250 (que é o código para “sucesso” no SMTP), é sinal de que a saudação foi bem-sucedida.

Iniciando a criptografia TLS

Se você estiver se conectando à porta 587 no servidor SMTP (ou seja, a criptografia TLS está sendo usada), será preciso chamar o método starttls() a seguir. Esse passo obrigatório habilita a criptografia em sua conexão. Se você

estiver se conectando à porta 465 (usando SSL), a criptografia já estará configurada e você deverá pular esse passo.

Eis um exemplo de uma chamada ao método starttls():

```
>>> smtpObj.starttls()
(220, b'2.0.0 Ready to start TLS')
```

starttls() coloca sua conexão SMTP em modo TLS. O 220 no valor de retorno informa que o servidor está pronto.

Fazendo login no servidor SMTP

Depois que sua conexão criptografada com o servidor SMTP estiver configurada, você poderá fazer login com seu nome de usuário (normalmente será o seu endereço de email) e a senha de email chamando o método login().

```
>>> smtpObj.login('my_email_address@gmail.com', 'MY_SECRET_PASSWORD')
(235, b'2.7.0 Accepted')
```

SENHAS ESPECÍFICAS DE APLICATIVOS DO GMAIL

O Gmail tem um recurso adicional de segurança para contas do Google chamado *senhas específicas de aplicativos*. Se receber uma mensagem de erro Application-specific password required (Senha específica do aplicativo necessária) quando seu programa tentar fazer login, você deverá configurar uma dessas senhas para o seu script Python. Dê uma olhada nos recursos em *http://nostarch.com/automatestuff/* para obter instruções detalhadas sobre como configurar uma senha específica de aplicativo para sua conta do Google.

Passe uma string com seu endereço de email como primeiro argumento e uma string com sua senha como o segundo argumento. O valor de retorno 235 indica que a autenticação foi bem-sucedida. O Python lançará uma exceção smtplib.SMTPAuthenticationError para senhas incorretas.

AVISO Tome cuidado ao inserir senhas em seu código-fonte. Se alguém copiar seu programa, essa pessoa terá acesso à sua conta de email! Chamar *input()* e fazer o usuário digitar a senha é uma boa ideia. Pode ser inconveniente ter de fornecer uma senha sempre que seu programa for executado, porém essa abordagem evitará que você deixe sua senha em um arquivo não criptografado em seu computador, onde um hacker ou um ladrão de laptops poderá obtê-la facilmente.

Enviando um email

Após ter feito login no servidor SMTP de seu provedor de emails, o método sendmail() poderá ser chamado para enviar o email. A chamada ao método sendmail() tem o seguinte aspecto:

```
>>> smtpObj.sendmail('my_email_address@gmail.com', 'recipient@example.com',
'Subject: So long.\nDear Alice, so long and thanks for all the fish. Sincerely, Bob')
{}
```

O método sendmail() exige três argumentos:

- seu endereço de email como uma string [para o endereço "from" (de) do email];
- o endereço de email do destinatário como uma string ou uma lista de strings para vários destinatários [para o endereço "to" (para)];
- o corpo do email como uma string.

O início da string com o corpo do email *deve* começar com 'Subject: \n' para a linha de assunto do email. O caractere '\n' de quebra de linha separa a linha de assunto do corpo principal do email.

O valor de retorno de sendmail() é um dicionário. Haverá um par de chave-valor no dicionário para cada destinatário a quem a entrega do email *falhar*. Um dicionário vazio indica que o email foi enviado *com sucesso* a todos os destinatários.

Desconectando-se do servidor SMTP

Não se esqueça de chamar o método quit() quando acabar de enviar os emails. Esse método desconectará o seu programa do servidor SMTP.

```
>>> smtpObj.quit()
(221, b'2.0.0 closing connection ko10sm23097611pbd.52 - gsmtp')
```

O valor de retorno 221 indica que a sessão foi encerrada.

Para rever todos os passos para se conectar e fazer login no servidor, enviar emails e desconectar, veja a seção "Enviando emails".

IMAP

Assim como o SMTP é o protocolo para enviar emails, o IMAP (*Internet Message Access Protocol*, ou Protocolo de acesso a mensagens da Internet) especifica o modo de se comunicar com um servidor de um provedor de emails e obter emails enviados ao seu endereço. O Python inclui um módulo imaplib, porém, na verdade, o módulo imapclient de terceiros é mais fácil de usar. Este capítulo oferece uma introdução ao uso do IMAPClient; a documentação completa está disponível em *http://imapclient.readthedocs.org/*.

O módulo imapclient faz download de emails de um servidor IMAP em um formato bem complexo. É mais provável que você vá querer converter esse formato em valores mais simples de string. O módulo pyzmail faz o trabalho pesado de parse dessas mensagens de email para você. A documentação completa do PyzMail pode ser encontrada em *http://www.magiksys.net/pyzmail/*.

Instale o imapclient e o pyzmail a partir de uma janela do Terminal. O apêndice A apresenta os passos para a instalação de módulos de terceiros.

Obtendo e apagando emails com o IMAP

Encontrar e obter um email em Python é um processo de vários passos que exige os módulos de terceiros imapclient e pyzmail. Somente para oferecer uma visão geral, eis um exemplo completo que inclui fazer login em um servidor IMAP, procurar os emails, acessá-los e extrair o texto das mensagens de email.

```
>>> import imapclient
>>> imapObj = imapclient.IMAPClient('imap.gmail.com', ssl=True)
>>> imapObj.login('my_email_address@gmail.com', 'MY_SECRET_PASSWORD')
'my_email_address@gmail.com Jane Doe authenticated (Success)'
>>> imapObj.select_folder('INBOX', readonly=True)
>>> UIDs = imapObj.search(['SINCE 05-Jul-2014'])
>>> UIDs
[40032, 40033, 40034, 40035, 40036, 40037, 40038, 40039, 40040, 40041]
>>> rawMessages = imapObj.fetch([40041], ['BODY[]', 'FLAGS'])
>>> import pyzmail
>>> message = pyzmail.PyzMessage.factory(rawMessages[40041]['BODY[]'])
>>> message.get_subject()
```

```
'Hello!'
>>> message.get_addresses('from')
[('Edward Snowden', 'esnowden@nsa.gov')]
>>> message.get_addresses('to')
[(Jane Doe', 'jdoe@example.com')]
>>> message.get_addresses('cc')
[]
>>> message.get_addresses('bcc')
[]
>>> message.text_part != None
True
>>> message.text_part.get_payload().decode(message.text_part.charset)
'Follow the money.\r\n\r\n-Ed\r\n'
>>> message.html_part != None
True
>>> message.html_part.get_payload().decode(message.html_part.charset)
'<div dir="ltr"><div>So long, and thanks for all the fish!<br><br></div>-Al<br>
</div>\r\n'
>>> imapObj.logout()
```

Não é necessário memorizar esses passos. Após descrevermos cada passo em detalhes, você poderá retornar a essa visão geral para refrescar sua memória.

Conectando-se a um servidor IMAP

Assim como precisamos de um objeto SMTP para fazer a conexão com um servidor SMTP e enviar emails, devemos ter um objeto IMAPClient para fazer a conexão com um servidor IMAP e receber emails. Inicialmente, você deverá ter o nome de domínio do servidor IMAP de seu provedor de emails. Esse será um nome diferente do nome de domínio do servidor SMTP. A tabela 16.2 lista os servidores IMAP de diversos provedores populares de email.

Tabela 16.2 – Provedores de email e seus servidores IMAP

Provedor	Nome de domínio do servidor IMAP
Gmail	*imap.gmail.com*
Outlook.com/Hotmail.com	*imap-mail.outlook.com*
Yahoo Mail	*imap.mail.yahoo.com*

AT&T	*imap.mail.att.net*
Comcast	*imap.comcast.net*
Verizon	*incoming.verizon.net*

De posse do nome de domínio do servidor IMAP, chame a função imapclient.IMAPClient() para criar um objeto IMAPClient. A maioria dos provedores de email exige criptografia SSL, portanto passe o argumento nomeado ssl=True. Digite o seguinte no shell interativo (usando o nome de domínio de seu provedor):

```
>>> import imapclient
>>> imapObj = imapclient.IMAPClient('imap.gmail.com', ssl=True)
```

Em todos os exemplos do shell interativo nas próximas seções, a variável imapObj conterá um objeto IMAPClient retornado pela função imapclient.IMAPClient(). Nesse contexto, um *cliente* é o objeto que se conecta ao servidor.

Fazendo login no servidor IMAP

Depois que tiver um objeto IMAPClient, chame o seu método login() passando-lhe o nome do usuário (geralmente é o seu endereço de email) e a senha como strings.

```
>>> imapObj.login('my_email_address@gmail.com', 'MY_SECRET_PASSWORD')
'my_email_address@gmail.com Jane Doe authenticated (Success)'
```

AVISO Lembre-se de jamais escrever uma senha diretamente em seu código! Em vez disso, crie seu programa para que ele aceite a senha retornada por input().

Se o servidor IMAP rejeitar essa combinação de nome de usuário/senha, o Python lançará uma exceção imaplib.error. Em contas Gmail, talvez seja necessário usar uma senha específica de aplicativo; para obter mais detalhes, consulte a seção “Senhas específicas de aplicativos do Gmail”.

Procurando emails

Depois de fazer login, obter um email em que você está interessado é um processo de dois passos. Inicialmente, selecione uma pasta em que você queira procurar. Em seguida, chame o método search() do objeto IMAPClient

passando-lhe uma string com as palavras-chaves de pesquisa do IMAP.

Selecionando uma pasta

Quase todas as contas têm uma pasta INBOX por padrão, mas você também poderá obter uma lista de pastas ao chamar o método list_folders() do objeto IMAPClient. Esse método retorna uma lista de tuplas. Cada tupla contém informações sobre uma única pasta. Prossiga com o exemplo no shell interativo digitando o seguinte:

```
>>> import pprint
>>> pprint.pprint(imapObj.list_folders())
[(('\\HasNoChildren',), '/', 'Drafts'),
(('\\HasNoChildren',), '/', 'Filler'),
(('\\HasNoChildren',), '/', 'INBOX'),
(('\\HasNoChildren',), '/', 'Sent'),
--trecho removido--
(('\\HasNoChildren', '\\Flagged'), '/', '[Gmail]/Starred'),
(('\\HasNoChildren', '\\Trash'), '/', '[Gmail]/Trash')]
```

Essa será a aparência de sua saída se você tiver uma conta Gmail. (O Gmail chama suas pastas de *labels*, mas elas funcionam da mesma maneira que as pastas.) Os três valores em cada tupla – por exemplo, (('\\HasNoChildren',), '/', 'INBOX') – são:

- Uma tupla com as flags da pasta. (O que essas flags representam exatamente está além do escopo deste livro, e você poderá ignorar esse campo sem que haja problemas.)
- O delimitador usado na string de nome para separar as pastas-pais e as subpastas.
- O nome completo da pasta.

Para selecionar uma pasta em que será feita a pesquisa, passe o nome da pasta como uma string ao método select_folder() do objeto IMAPClient.

```
>>> imapObj.select_folder('INBOX', readonly=True)
```

Você poderá ignorar o valor de retorno de select_folder(). Se a pasta selecionada não existir, o Python gerará uma exceção imaplib.error.

O argumento nomeado readonly=True impede que você faça alterações ou

remova emails acidentalmente dessa pasta durante as chamadas subsequentes de métodos. A menos que você *queira* apagar emails, sempre configurar readonly com True é uma boa ideia.

Realizando a pesquisa

Com uma pasta selecionada, podemos agora procurar emails usando o método search() do objeto IMAPClient. O argumento de search() é uma lista de strings, cada qual formatada com as chaves de pesquisa do IMAP. A tabela 16.3 descreve as diversas chaves de pesquisa.

Tabela 16.3 – Chaves de pesquisa do IMAP

Chave de pesquisa	Significado
'ALL'	Retorna todas as mensagens da pasta. Você poderá se deparar com limites de tamanho no imaplib se solicitar todas as mensagens de uma pasta extensa. Consulte a seção "Limites de tamanho".
'BEFORE *data*', 'ON *data*', 'SINCE *data*'	Essas três chaves de pesquisa retornam, respectivamente, as mensagens recebidas pelo servidor IMAP antes da *data* especificada, na *data* especificada e após essa *data*. A data deve estar formatada como 05-Jul-2015. Além disso, enquanto 'SINCE 05-Jul-2015' corresponderá às mensagens de 5 de julho e após essa data, 'BEFORE 05-Jul-2015' corresponderá somente às mensagens anteriores a 5 de julho, mas sem incluir essa data.
'SUBJECT *string*', 'BODY *string*', 'TEXT *string*'	Retorna mensagens em que *string* está presente no assunto, no corpo ou em um deles, respectivamente. Se *string* tiver espaços, insira aspas duplas ao seu redor: 'TEXT "search with spaces"'.
'FROM *string*', 'TO *string*', 'CC *string*', 'BCC *string*'	Retorna todas as mensagens em que *string* está presente no endereço de email em "from" (de), nos endereços em "to" (para), nos endereços em "cc" (carbon copy, ou cópia) ou nos endereços em "bcc" (blind carbon copy, ou cópia oculta), respectivamente. Se houver vários endereços de email em *string*, separe-os com espaços e inclua aspas duplas ao redor deles: 'CC "*firstcc@example.com secondcc@example.com*"'.
'SEEN', 'UNSEEN'	Retorna todas as mensagens com e sem a flag *\Seen*, respectivamente. Um email terá a flag *\Seen* se ela for acessada com uma chamada de método fetch() (descrita posteriormente) ou se for clicada quando você estiver verificando seus emails em um programa de emails ou no navegador web. É mais comum dizer que o email foi "lido" (read) em

	vez de “visto” (seen), mas ambos têm o mesmo significado.
'ANSWERED', 'UNANSWERED'	Retorna todas as mensagens com e sem a flag *\Answered*, respectivamente. Uma mensagem terá a flag *\Answered* se ela tiver sido respondida.
'DELETED', 'UNDELETED'	Retorna todas as mensagens com e sem a flag *\Deleted*, respectivamente. As mensagens de email apagadas com o método delete_messages() recebem a flag *\Deleted*, porém não serão apagadas permanentemente até que o método expunge() seja chamado (veja a seção “Apagando emails”). Observe que alguns provedores de emails como o Gmail expurgam os emails automaticamente.
'DRAFT', 'UNDRAFT'	Retorna todas as mensagens com e sem a flag *\Draft*, respectivamente. As mensagens de rascunho (draft) normalmente são mantidas em uma pasta Drafts separada, e não na pasta INBOX.
'FLAGGED', 'UNFLAGGED'	Retorna todas as mensagens com e sem a flag *\Flagged*, respectivamente. Essa flag normalmente é usada para marcar mensagens de email como “Importante” (Important) ou “Urgente” (Urgent).
'LARGER *N*', 'SMALLER *N*'	Retorna todas as mensagens com maiores ou menores do que *N* bytes, respectivamente.
'NOT *chave-de-pesquisa*'	Retorna as mensagens que *não* teriam sido retornadas com a *chave-de-pesquisa*.
'OR *chave-de-pesquisa1 chave-de-pesquisa2*'	Retorna as mensagens que correspondam à primeira *ou* à segunda *chave-de-pesquisa*.

Observe que alguns servidores IMAP podem ter implementações um pouco diferentes em relação ao modo como tratam suas flags e as chaves de pesquisa. Talvez seja necessário realizar alguns experimentos no shell interativo para ver exatamente como eles se comportam.

Podemos passar várias strings com chaves de pesquisa IMAP na lista de argumentos ao método search(). As mensagens retornadas serão aquelas que corresponderem a *todas* as chaves de pesquisa. Se quiser fazer a correspondência com *qualquer* chave de pesquisa, utilize a chave de pesquisa OR. Para as chaves de pesquisa NOT e OR, uma e duas chaves de pesquisa completas devem estar após NOT e OR, respectivamente.

Eis alguns exemplos de chamadas ao método search(), juntamente com seus

significados:

- **imapObj.search(['ALL'])** Retorna todas as mensagem da pasta selecionada no momento.
- **imapObj.search(['ON 05-Jul-2015'])** Retorna todas as mensagens enviadas em 5 de julho de 2015.
- **imapObj.search(['SINCE 01-Jan-2015', 'BEFORE 01-Feb-2015', 'UNSEEN'])** Retorna todas as mensagens enviadas em janeiro de 2015 que não tenham sido lidas. (Observe que isso quer dizer *no* dia 1 de janeiro e *depois* dessa data, até o dia 1 de fevereiro, *mas sem incluir* essa data.)
- **imapObj.search(['SINCE 01-Jan-2015', 'FROM alice@example.com'])** Retorna todas as mensagens de *alice@example.com* enviadas desde o início de 2015.
- **imapObj.search(['SINCE 01-Jan-2015', 'NOT FROM alice@example.com'])** Retorna todas as mensagens enviadas por todos, exceto por *alice@example.com*, desde o início de 2015.
- **imapObj.search(['OR FROM alice@example.com FROM bob@example.com'])** Retorna todas as mensagens enviadas por *alice@example.com* ou por *bob@example.com*.
- **imapObj.search(['FROM alice@example.com', 'FROM bob@example.com'])** Exemplo capcioso! Essa pesquisa jamais retornará nenhuma mensagem, pois as mensagens devem corresponder a *todas* as chaves de pesquisa. Como pode haver somente um endereço “from” (de), será impossível que uma mensagem seja tanto de *alice@example.com* quanto de *bob@example.com*.

O método search() não retorna os emails propriamente ditos, mas IDs únicos (UIDs) de emails na forma de valores inteiros. Então você poderá passar esses UIDs ao método fetch() para obter o conteúdo do email.

Prossiga com o exemplo no shell interativo digitando o seguinte:

```
>>> UIDs = imapObj.search(['SINCE 05-Jul-2015'])
>>> UIDs
[40032, 40033, 40034, 40035, 40036, 40037, 40038, 40039, 40040, 40041]
```

Nesse caso, a lista de IDs das mensagens (para as mensagens recebidas a

partir de 5 de julho) retornada por search() é armazenada em UIDs. A lista de UIDs retornada em seu computador será diferente daquela mostrada aqui; os UIDs são únicos para uma conta de email em particular. Ao passar os UIDs posteriormente a outras chamadas de função, utilize os valores de UID recebidos, e não os que foram mostrados nos exemplos deste livro.

Limites de tamanho

Se sua pesquisa corresponder a uma grande quantidade de mensagens de email, o Python poderá gerar uma exceção com a mensagem imaplib.error: got more than 10000 bytes (imaplib.error: mais de 10.000 bytes obtidos). Quando isso ocorrer, você deverá se desconectar e conectar-se novamente ao servidor IMAP para tentar de novo.

Esse limite foi definido para evitar que seus programas Python consumam muita memória. Infelizmente, o limite default de tamanho geralmente é baixo demais. Esse limite pode ser alterado de 10.000 bytes para 10.000.000 bytes ao executar o código a seguir:

```
>>> import imaplib
>>> imaplib._MAXLINE = 10000000
```

Isso evitará que essa mensagem de erro apareça novamente. Você poderá incluir essas duas linhas como parte de todo programa IMAP que você criar.

USANDO O MÉTODO GMAIL_SEARCH( ) DE IMAPCLIENT

Se você estiver fazendo login no servidor *imap.gmail.com* para acessar uma conta Gmail, o objeto IMAPClient disponibilizará uma função extra de pesquisa que imita a barra de pesquisa na parte superior da página web do Gmail, que aparece em destaque na figura 16.1.

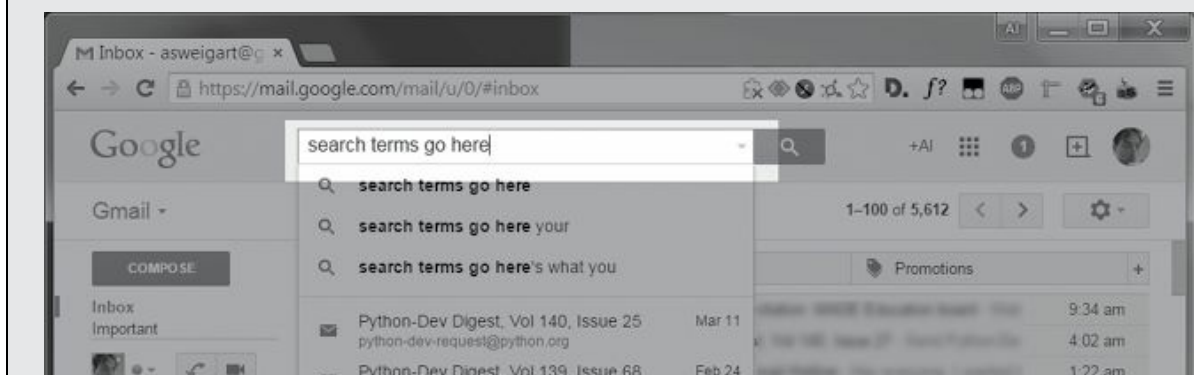
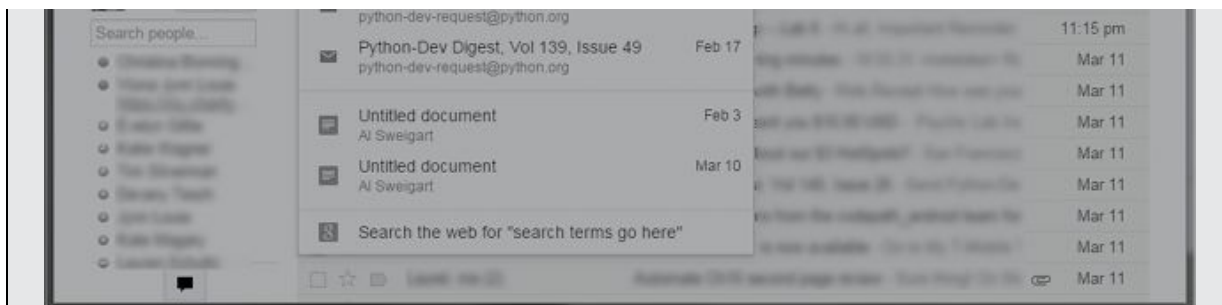



Figura 16.1 – A barra de pesquisa na parte superior da página web do Gmail.

Em vez de fazer pesquisas com as chaves de pesquisa do IMAP, você poderá usar a ferramenta de pesquisa do Gmail, que é mais sofisticada. O Gmail faz um bom trabalho de correspondência de palavras aproximadas (por exemplo, uma pesquisa pelo termo “driving” fará também a correspondência de “drive” e de “drove”) e de ordenação dos resultados da pesquisa de acordo com as correspondências mais significativas. Os operadores avançados de pesquisa do Gmail também podem ser utilizados (acesse *http://nostarch.com/automatestuff/* para obter mais informações). Se você fizer login em uma conta Gmail, passe os termos de pesquisa ao método gmail_search() em vez de passá-los ao método search(), como no exemplo a seguir no shell interativo:

```
>>> UIDs = imapObj.gmail_search('meaning of life')
>>> UIDs
[42]
```

Ah, sim, *aqui está* aquele email com o significado da vida (meaning of life)! Eu estava procurando-o.

Buscando um email e marcando-o como lido

De posse de uma lista de UIDs, você poderá chamar o método fetch() do objeto IMAPClient para obter o conteúdo propriamente dito dos emails.

A lista de UIDs será o primeiro argumento de fetch(). O segundo argumento deverá ser a lista ['BODY[]'], que diz a fetch() para fazer o download de todo o conteúdo do corpo dos emails especificados em sua lista de UIDs.

Vamos continuar com o nosso exemplo no shell interativo:

```
>>> rawMessages = imapObj.fetch(UIDs, ['BODY[]'])
>>> import pprint
>>> pprint.pprint(rawMessages)
{40040: {'BODY[]': 'Delivered-To: my_email_address@gmail.com\r\n'
```

```
                'Received: by 10.76.71.167 with SMTP id '
--trecho removido--
                '\r\n'
                '------=_Part_6000970_707736290.1404819487066--\r\n',
        'SEQ': 5430}}
```

Importe pprint e passe o valor de retorno de fetch(), armazenado na variável rawMessages, a pprint.pprint() para efetuar um "pretty print" (apresentação elegante), e você verá que esse valor de retorno é um dicionário aninhado de mensagens em que os UIDs são as chaves. Cada mensagem é armazenada como um dicionário com duas chaves: 'BODY[]' e 'SEQ'. A chave 'BODY[]' é mapeada ao corpo do email. A chave 'SEQ' corresponde a um *número sequencial*, que tem uma função semelhante à de UID. Você poderá ignorar esse valor sem que haja problemas.

Como podemos ver, o conteúdo da mensagem na chave 'BODY[]' é bastante ininteligível. Ele está em um formato chamado RFC 822, projetado para ser lido pelos servidores IMAP. Contudo não é necessário entender o formato RFC 822; mais adiante neste capítulo, o módulo pyzmail o interpretará para você.

Ao selecionar uma pasta para pesquisar, você chamou select_folder() com o argumento nomeado readonly=True. Fazer isso impede que você apague um email acidentalmente – mas significa também que os emails não serão marcados como lidos se forem acessados com o método fetch(). Se *quiser* que os emails sejam marcados como lidos quando forem acessados, passe readonly=False para select_folder(). Se a pasta selecionada já estiver em modo somente de leitura, você poderá selecionar novamente a pasta atual por meio de outra chamada a select_folder(), desta vez com o argumento nomeado readonly=False:

```
>>> imapObj.select_folder('INBOX', readonly=False)
```

Obtendo endereços de email de uma mensagem pura

As mensagens puras retornadas pelo método fetch() ainda não são muito úteis às pessoas que queiram somente ler seus emails. O módulo pyzmail faz parse dessas mensagens puras e as retorna como objetos PyzMessage, o que torna o assunto (subject), o corpo (body), o campo "Para" (To), o campo "De"

(From) e outras partes do email facilmente acessíveis ao seu código Python.

Prossiga com o exemplo no shell interativo fazendo o seguinte (use os UIDs de sua própria conta de email, e não aqueles mostrados aqui):

```
>>> import pyzmail
>>> message = pyzmail.PyzMessage.factory(rawMessages[40041]['BODY[]'])
```

Inicialmente, importe pyzmail. Em seguida, para criar um objeto PyzMessage para um email, chame a função pyzmail.PyzMessage.factory() e passe-lhe a seção 'BODY[]' da mensagem pura. Armazene o resultado em message. Agora message contém um objeto PyzMessage, que tem diversos métodos para facilitar a obtenção da linha de assunto do email bem como do endereço de quem o enviou e os endereços dos destinatários. O método get_subject() retorna o assunto na forma de um valor simples de string. O método get_addresses() retorna uma lista de endereços para o campo que for passado para ele. Por exemplo, as chamadas a esses métodos terão o seguinte aspecto:

```
>>> message.get_subject()
'Hello!'
>>> message.get_addresses('from')
[('Edward Snowden', 'esnowden@nsa.gov')]
>>> message.get_addresses('to')
[(Jane Doe', 'my_email_address@gmail.com')]
>>> message.get_addresses('cc')
[]
>>> message.get_addresses('bcc')
[]
```

Observe que o argumento de get_addresses() pode ser 'from', 'to', 'cc' ou 'bcc'. O valor de retorno de get_addresses() é uma lista de tuplas. Cada tupla contém duas strings: a primeira corresponde ao nome associado ao endereço de email e a segunda é o endereço de email propriamente dito. Se não houver nenhum endereço no campo solicitado, get_addresses() retornará uma lista vazia. Nesse exemplo, os campos 'cc' para cópia e 'bcc' para cópia oculta não contêm nenhum endereço; sendo assim, retornaram listas vazias.

Obtendo o corpo de uma mensagem pura

Os emails podem ser enviados em formato texto simples, HTML ou em

ambos os formatos. Os emails em formato texto simples contêm somente texto, enquanto os emails HTML podem ter cores, fontes, imagens e outros recursos que deixam a mensagem de email semelhante a uma pequena página web. Se um email tiver somente texto simples, seu objeto PyzMessage terá o atributo html_part definido com None. De modo semelhante, se um email tiver somente HTML, seu objeto PyzMessage terá o atributo text_part definido com None.

Caso contrário, o valor de text_part ou de html_part terá um método get_payload() que retornará o corpo do email como um valor do tipo de dado *bytes*. (O tipo de dado bytes está além do escopo deste livro.) Porém esse dado *ainda* não é um valor de string que possamos usar. Argh! O último passo consiste em chamar o método decode() no valor em bytes retornado por get_payload(). O método decode() aceita um argumento: a codificação de caracteres da mensagem, armazenada no atributo text_part.charset ou html_part.charset. Esse método, finalmente, retornará a string com o corpo do email.

Prossiga com o exemplo no shell interativo digitando o seguinte:

```
u >>> message.text_part != None
  True
  >>> message.text_part.get_payload().decode(message.text_part.charset)
v 'So long, and thanks for all the fish!\r\n\r\n-Al\r\n'
w >>> message.html_part != None
  True
x >>> message.html_part.get_payload().decode(message.html_part.charset)
  '<div dir="ltr"><div>So long, and thanks for all the fish!<br><br></div>-Al<br>
  </div>\r\n'
```

O email com o qual estamos trabalhando tem conteúdo tanto em formato texto simples quanto em HTML, portanto o objeto PyzMessage armazenado em message tem atributos text_part e html_part diferentes de None u w. Chamar get_payload() no text_part da mensagem e, em seguida, chamar decode() no valor em bytes fará uma string com a versão em texto do email ser retornada v. Usar get_payload() e decode() no html_part da mensagem fará uma string com a versão HTML do email ser retornada x.

Apagando emails

Para apagar emails, passe uma lista de UIDs de mensagens ao método delete_messages() do objeto IMAPClient. Isso faz os emails serem marcados com a flag *\Deleted*. Chamar o método expunge() fará todos os emails com a flag *\Deleted* serem permanentemente apagados da pasta selecionada no momento. Considere o exemplo a seguir no shell interativo:

```
u >>> imapObj.select_folder('INBOX', readonly=False)
v >>> UIDs = imapObj.search(['ON 09-Jul-2015'])
  >>> UIDs
  [40066]
  >>> imapObj.delete_messages(UIDs)
w {40066: ('\\Seen', '\\Deleted')}
  >>> imapObj.expunge()
  ('Success', [(5452, 'EXISTS')])
```

Nesse caso, selecionamos o inbox chamando select_folder() no objeto IMAPClient, passando-lhe 'INBOX' como o primeiro argumento; também passamos o argumento nomeado readonly=False para que pudéssemos apagar os emails u. Pesquisamos o inbox em busca de mensagens recebidas em uma data específica e armazenamos os IDs das mensagens retornadas em UIDs v. Chamar delete_message() passando UIDs faz um dicionário ser retornado; cada par chave-valor contém um ID de mensagem e uma tupla com as flags dessa mensagem, que agora deverá incluir *\Deleted* w. Chamar expunge() apagará permanentemente as mensagens com a flag *\Deleted* e retornará uma mensagem de sucesso se não houver nenhum problema na expurgação dos emails. Observe que alguns provedores de emails, por exemplo, o Gmail, fazem automaticamente a expurgação dos emails apagados com delete_messages() em vez de esperar um comando de expurgação do cliente IMAP.

Desconectando-se do servidor IMAP

Quando seu programa acabar de obter ou de apagar os emails, basta chamar o método logout() de IMAPClient para se desconectar do servidor IMAP.

```
>>> imapObj.logout()
```

Se o seu programa executar durante alguns minutos ou mais, poderá ocorrer um *timeout* no servidor IMAP, ou seja, ele será automaticamente desconectado. Nesse caso, a próxima chamada de método que seu programa fizer no objeto IMAPClient gerará uma exceção como esta:

```
imaplib.abort: socket error: [WinError 10054] An existing connection was forcibly
closed by the remote host
```

Caso isso ocorra, seu programa deverá chamar imapclient.IMAPClient() para se conectar novamente.

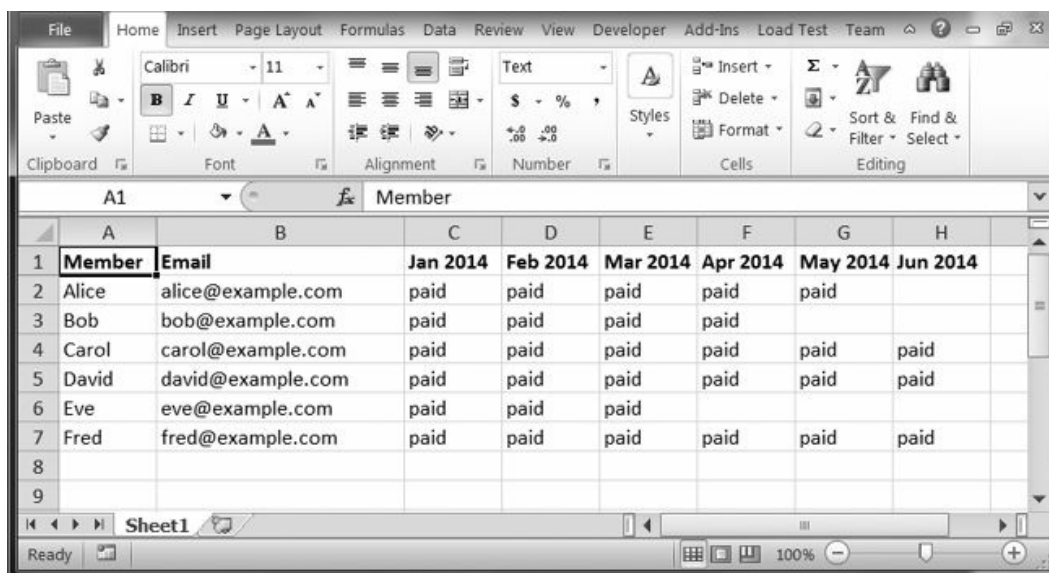
Ufa! É isso. Houve vários obstáculos a serem vencidos, porém agora você tem uma maneira de fazer seus programas Python efetuarem login em uma conta de email e buscar os emails. Você sempre poderá dar uma olhada na visão geral apresentada na seção "Obtendo e apagando emails com o IMAP" quando houver necessidade de relembrar todos os passos.

Projeto: Enviando emails com aviso de vencimento de pagamento

Suponha que você tenha se oferecido como "voluntário" para monitorar as datas de vencimento de pagamento dos sócios do Mandatory Volunteerism Club (Clube de voluntariado obrigatório). Essa é uma tarefa realmente maçante, que envolve manter uma planilha com os nomes de todos que fizeram o pagamento em cada mês e enviar emails de lembrete àqueles que não pagaram. Em vez de percorrer a planilha por conta própria, copiando e colando o mesmo email para todos que estiverem com os pagamentos atrasados – você adivinhou –, crie um script que faça isso por você.

De modo geral, o seu programa deverá:

- Ler dados de uma planilha Excel.
- Identificar todos os sócios que não fizeram seu pagamento no último mês.
- Localizar seus endereços de email e enviar-lhes lembretes personalizados.

Isso significa que o seu código deverá fazer o seguinte:

- Abrir e ler as células de um documento Excel com o módulo openpyxl. (Veja o capítulo 12 para saber como trabalhar com arquivos Excel.)

• Criar um dicionário com os sócios que estiverem com os pagamentos atrasados.
• Fazer login em um servidor SMTP chamando smtplib.SMTP(), ehlo(), starttls() e login().
• Para todos os sócios que estiverem com os pagamentos atrasados, enviar um email personalizado de lembrete chamando o método sendmail().

Abra uma nova janela no editor de arquivo e salve o programa como *sendDuesReminders.py*.

Passo 1: Abrir o arquivo Excel

Vamos supor que a planilha Excel a ser usada para monitorar os pagamentos dos sócios tenha a aparência mostrada na figura 16.2 e esteja em um arquivo chamado *duesRecords.xlsx*. O download desse arquivo pode ser feito a partir de *http://nostarch.com/automatestuff/*.

Member	Email	Jan 2014	Feb 2014	Mar 2014	Apr 2014	May 2014	Jun 2014
Alice	alice@example.com	paid	paid	paid	paid	paid	
Bob	bob@example.com	paid	paid	paid	paid		
Carol	carol@example.com	paid	paid	paid	paid	paid	paid
David	david@example.com	paid	paid	paid	paid	paid	paid
Eve	eve@example.com	paid	paid	paid			
Fred	fred@example.com	paid	paid	paid	paid	paid	paid

Figura 16.2 – A planilha para monitorar os pagamentos dos sócios.

Essa planilha contém o nome e o endereço de email de todos os sócios. Cada mês tem uma coluna associada para administrar o status de pagamento dos sócios. A célula correspondente a cada sócio será marcada com o texto *paid* (pago) depois que eles fizerem seus pagamentos.

O programa deverá abrir o arquivo *duesRecords.xlsx* e descobrir qual é a coluna referente ao último mês chamando o método get_highest_column().

(Você pode consultar o capítulo 12 para obter mais informações sobre como acessar as células em arquivos de planilhas Excel com o módulo openpyxl.) Digite o seguinte na janela do editor de arquivo:

```
   #! python3
   # sendDuesReminders.py – Envia emails de acordo com o status de pagamento na
   planilha.

   import openpyxl, smtplib, sys

   # Abre a planilha e obtém o status do último pagamento.
u wb = openpyxl.load_workbook('duesRecords.xlsx')
v sheet = wb.get_sheet_by_name('Sheet1')

w lastCol = sheet.get_highest_column()
x latestMonth = sheet.cell(row=1, column=lastCol).value

   # TODO: Verifica o status de pagamento de cada sócio.

   # TODO: Faz login na conta de email.

   # TODO: Envia emails de lembrete.
```

Após importar os módulos openpyxl, smtplib e sys, abrimos o arquivo *duesRecords.xlsx* e armazenamos o objeto Workbook resultante em wb u. Em seguida, obtivemos Sheet 1 e armazenamos o objeto Worksheet resultante em sheet v. Agora que temos um objeto Worksheet, podemos acessar as linhas, as colunas e as células. Armazenamos a coluna de maior número em lastCol w e usamos o número de linha 1 e lastCol para acessar a célula que deverá armazenar o dado do mês mais recente. Obtivemos o valor dessa célula e o armazenamos em latestMonth x.

Passo 2: Localizar todos os sócios que não fizeram o pagamento

Após ter determinado o número da coluna referente ao último mês (armazenado em lastCol), podemos percorrer todas as linhas após a primeira (que contém os cabeçalhos da coluna) em um loop para ver quais sócios têm o texto *paid* (pago) na célula referente ao pagamento desse mês. Se o sócio

não tiver pagado, você poderá obter o seu nome e o seu endereço de email nas colunas 1 e 2, respectivamente. Essas informações serão inseridas no dicionário unpaidMembers, que manterá um registro de todos os sócios que ainda não pagaram no último mês. Adicione o código a seguir em *sendDuesReminder.py*.

```
#! python3
# sendDuesReminders.py – Envia emails de acordo com o status de pagamento na
planilha.

--trecho removido--

# Verifica o status de pagamento de cada sócio.
unpaidMembers = {}
u for r in range(2, sheet.get_highest_row() + 1):
v     payment = sheet.cell(row=r, column=lastCol).value
      if payment != 'paid':
w         name = sheet.cell(row=r, column=1).value
x         email = sheet.cell(row=r, column=2).value
y         unpaidMembers[name] = email
```

Esse código cria um dicionário unpaidMembers vazio e, em seguida, percorre todas as linhas após a primeira em um loop u. Para cada linha, o valor na coluna mais recente será armazenado em payment v. Se payment não for igual a 'paid', o valor da primeira coluna será armazenado em name w, o valor da segunda coluna será armazenado em email x e name e email serão adicionados a unpaidMembers y.

Passo 3: Enviar emails personalizados para servir de lembrete

Depois que você tiver uma lista com todos os sócios que não fizeram o pagamento, será hora de enviar emails que servirão de lembrete a eles. Adicione o código a seguir em seu programa, mas utilize informações contendo o seu endereço de email e o provedor reais:

```
#! python3
# sendDuesReminders.py – Envia emails de acordo com o status de pagamento na
planilha.
```

```
--trecho removido--

# Faz login na conta de email.
smtpObj = smtplib.SMTP('smtp.gmail.com', 587)
smtpObj.ehlo()
smtpObj.starttls()
smtpObj.login('my_email_address@gmail.com', sys.argv[1])
```

Crie um objeto SMTP ao chamar smtplib.SMTP() e passe-lhe o nome de domínio e a porta de seu provedor. Chame ehlo() e starttls(); em seguida, chame login() passando seu endereço de email e sys.argv[1], que terá sua string de senha armazenada. Forneça a senha como um argumento da linha de comando sempre que executar o programa para evitar que sua senha fique registrada em seu código-fonte.

Depois que seu programa fizer login em sua conta de email, ele deverá percorrer o dicionário unpaidMembers e enviar um email personalizado para o endereço de email de cada sócio. Acrescente o código a seguir em *sendDuesReminders.py*:

```
#! python3
# sendDuesReminders.py – Envia emails de acordo com o status de pagamento na
planilha.

--trecho removido--

# Envia emails de lembrete.
for name, email in unpaidMembers.items():
u    body = "Subject: %s dues unpaid.\nDear %s,\nRecords show that you have not
paid dues for %s. Please make this payment as soon as possible. Thank you!'" %
(latestMonth, name, latestMonth)
v    print('Sending email to %s...' % email)
w    sendmailStatus = smtpObj.sendmail('my_email_address@gmail.com', email, body)

x    if sendmailStatus != {}:
         print('There was a problem sending email to %s: %s' % (email,
sendmailStatus))
smtpObj.quit()
```

Esse código percorre os nomes e os emails em unpaidMembers em um loop.

Para cada sócio que não tenha efetuado o pagamento, criaremos uma mensagem personalizada com o último mês e o nome do sócio e armazenaremos essa mensagem em body u. Exibimos uma saída informando que estamos enviando um email ao endereço de email desse sócio v. Em seguida, chamamos sendmail(), passando-lhe o endereço de quem está enviando o email e a mensagem personalizada w. Armazenamos o valor de retorno em sendmailStatus.

Lembre-se de que o método sendmail() retornará um valor de dicionário diferente de vazio se o servidor SMTP informar que houve um erro ao enviar esse email em particular. A última parte do loop for em x verifica se o dicionário retornado é diferente de vazio e, em caso afirmativo, o endereço de email do destinatário e o dicionário retornado serão exibidos.

Depois que o programa acabar de enviar todos os emails, o método quit() será chamado para encerrar a conexão com o servidor SMTP.

Ao executar o programa, a saída será semelhante a:

```
Sending email to alice@example.com...
Sending email to bob@example.com...
Sending email to eve@example.com...
```

Os destinatários receberão um email semelhante ao da figura 16.3.

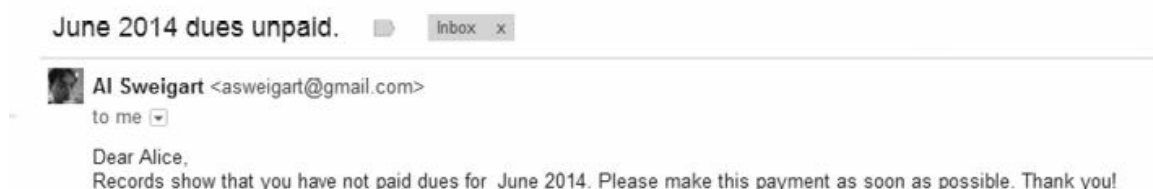



Figura 16.3 – Um email enviado automaticamente por sendDuesReminders.py.

Enviando mensagens de texto com o Twilio

É mais provável que a maioria das pessoas esteja mais perto de seus telefones do que de seus computadores; sendo assim, as mensagens de texto podem ser uma maneira mais imediata e confiável de enviar notificações do que o email. Além disso, o fato de as mensagens de texto serem breves aumentará as chances de uma pessoa lê-las.

Nesta seção, veremos como se inscrever para usar o serviço gratuito do Twilio e utilizar seu módulo Python para enviar mensagens de texto. O Twilio é um *serviço de gateway SMS* – isso significa que ele é um serviço que possibilita enviar mensagens de texto a partir de seus programas. Embora haja um limite em relação à quantidade de mensagens de textos que possa ser enviada por mês e o texto seja prefixado com as palavras *Sent from a Twilio trial account* (Enviado a partir de uma conta trial do Twilio), esse serviço trial provavelmente será adequado para seus programas pessoais. Não há prazo para usar o trial gratuito; não será necessário fazer um upgrade para uma versão paga posteriormente.

O Twilio não é o único serviço de gateway SMS existente. Se preferir não usar o Twilio, você poderá encontrar serviços alternativos por meio de pesquisas online em busca de *free sms gateway* (gateway sms gratuito), *python sms api* ou até mesmo *twilio alternatives* (alternativas ao twilio).

Antes de se inscrever para criar uma conta no Twilio, instale o módulo twilio. O apêndice A contém mais detalhes sobre a instalação de módulos de terceiros.

NOTA Esta seção é específica para os Estados Unidos. O Twilio oferece serviços de SMS para outros países que não sejam os Estados Unidos, porém essas especificidades não serão discutidas neste livro. Contudo o módulo twilio e suas funções operam da mesma maneira fora dos Estados Unidos. Acesse o site *http://twilio.com/* para obter mais informações.

Criando uma conta no Twilio

Acesse *http://twilio.com/* e preencha o formulário de inscrição. Após ter criado uma nova conta, será necessário fazer a verificação de um número de telefone celular para o qual você queira enviar mensagens de texto. (Essa verificação é necessária para evitar que as pessoas usem o serviço e enviem mensagens de texto spams a números de telefone quaisquer.)

Após receber o texto com o número de verificação, forneça-o no site do Twilio para provar que você é dono do telefone celular que está sendo verificado. Agora você poderá enviar mensagens de texto para esse número de telefone usando o módulo twilio.

O Twilio disponibiliza um número de telefone à sua conta trial, que será

usado para enviar as mensagens de texto. Outras duas informações serão necessárias: o SID de sua conta e o token de autenticação (auth). Essas informações podem ser encontradas na página Dashboard (Painel de controle) quando você fizer login em sua conta no Twilio. Esses valores atuarão como o seu nome de usuário e a senha no Twilio quando você fizer login a partir de um programa Python.

Enviando mensagens de texto

Após ter instalado o módulo twilio, criado uma conta no Twilio, efetuado a verificação de seu número de telefone, registrado um número de telefone no Twilio e obtido o SID e o token de autenticação de sua conta, finalmente você estará pronto para enviar mensagens de texto a você mesmo a partir de seus scripts Python.

Quando comparado a todos os passos para registrar a sua conta, podemos dizer que o código Python propriamente dito é bem simples. Com seu computador conectado à Internet, digite o seguinte no shell interativo, substituindo os valores das variáveis accountSID, authToken, myTwilioNumber e myCellPhone por suas informações reais:

```
u >>> from twilio.rest import TwilioRestClient
  >>> accountSID = 'ACxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx'
  >>> authToken  = 'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx'
v >>> twilioCli = TwilioRestClient(accountSID, authToken)
  >>> myTwilioNumber = '+14955551234'
  >>> myCellPhone = '+14955558888'
w >>> message = twilioCli.messages.create(body='Mr. Watson - Come here - I
want   to see you.', from_=myTwilioNumber, to=myCellPhone)
```

Alguns instantes após ter digitado a última linha, você deverá receber uma mensagem de texto em que se lê: *Sent from your Twilio trial account - Mr. Watson - Come here - I want to see you* (Enviado a partir de sua conta trial do Twilio - sr. Watson - Venha até aqui - Gostaria de vê-lo).

Por causa da maneira como o módulo twilio está configurado, será preciso importá-lo usando from twilio.rest import TwilioRestClient, e não apenas import twilio u. Armazene o SID de sua conta em accountSID e seu token de autenticação em authToken; em seguida, chame TwilioRestClient() e passe-lhe

accountSID e authToken como argumentos. A chamada a TwilioRestClient() retorna um objeto TwilioRestClient v. Esse objeto tem um atributo messages, que, por sua vez, tem um método create(); esse método pode ser usado para enviar suas mensagens de texto. Esse é o método que instruirá os servidores do Twilio a enviar sua mensagem de texto. Após armazenar seu número no Twilio e o número do telefone celular em myTwilioNumber e em myCellPhone, respectivamente, chame create() e passe-lhe argumentos nomeados para especificar o corpo da mensagem de texto, o número que está enviando a mensagem (myTwilioNumber) e o número do destinatário (myCellPhone) w.

O objeto Message retornado pelo método create() conterá informações sobre a mensagem de texto enviada. Prossiga com o exemplo no shell interativo digitando o seguinte:

```
>>> message.to
'+14955558888'
>>> message.from_
'+14955551234'
>>> message.body
'Mr. Watson - Come here - I want to see you.'
```

Os atributos to, from_ e body devem conter o número de seu telefone celular, o número do Twilio e a mensagem, respectivamente. Observe que o número de telefone que enviou a mensagem está no atributo from_ – com um underscore no final – e não em from. Isso ocorre porque from é uma palavra reservada em Python (você já a viu sendo usada na forma from modulename import * da instrução import, por exemplo), portanto ela não pode ser usada como um nome de atributo. Prossiga com o exemplo no shell interativo digitando o seguinte:

```
>>> message.status
'queued'
>>> message.date_created
datetime.datetime(2015, 7, 8, 1, 36, 18)
>>> message.date_sent == None
True
```

O atributo status deve fornecer uma string. Os atributos date_created e date_sent devem fornecer um objeto datetime se a mensagem for criada e

enviada. Pode parecer estranho que o atributo status esteja definido com 'queued' e que o atributo date_sent esteja definido com None quando você já tiver recebido a mensagem de texto. Isso ocorre porque o objeto Message foi capturado na variável message *antes* de o texto ter sido realmente enviado. Será preciso acessar novamente o objeto Message para ver os valores mais recentes de status e de date_sent. Toda mensagem do Twilio tem um ID único em forma de string (SID) que pode ser usado para acessar as últimas atualizações no objeto Message. Prossiga com o exemplo no shell interativo digitando o seguinte:

```
>>> message.sid
'SM09520de7639ba3af137c6fcb7c5f4b51'
u >>> updatedMessage = twilioCli.messages.get(message.sid)
>>> updatedMessage.status
'delivered'
>>> updatedMessage.date_sent
datetime.datetime(2015, 7, 8, 1, 36, 18)
```

Digitar message.sid exibe o SID longo dessa mensagem. Ao passar esse SID ao método get() do cliente Twilio u, um novo objeto Message com informações atualizadas será obtido. Nesse novo objeto Message, os atributos status e date_sent estarão corretos.

O atributo status estará definido com um dos seguintes valores de string: 'queued', 'sending', 'sent', 'delivered', 'undelivered' ou 'failed'. Esses valores de status são autoexplicativos, porém, para obter detalhes mais específicos, dê uma olhada nos recursos em *http://nostarch.com/automatestuff/*.

RECEBENDO MENSAGENS DE TEXTO COM O PYTHON

Infelizmente, receber mensagens de texto com o Twilio é um pouco mais complicado do que enviá-las. O Twilio exige que você tenha um site executando sua própria aplicação web. Esse assunto está além do escopo deste livro, mas você poderá encontrar mais detalhes nos recursos disponibilizados para este livro em *http://nostarch.com/automatestuff/*.

Projeto: Módulo “Envie uma mensagem a mim mesmo”

A pessoa a quem você enviará mensagens de texto com mais frequência a partir de seus programas provavelmente será você mesmo. Enviar mensagens de texto é uma ótima maneira de enviar notificações a si mesmo quando você estiver distante de seu computador. Se você automatizou uma tarefa maçante com um programa que demore algumas horas para ser executado, poderá fazê-lo enviar uma mensagem de texto avisando você que a tarefa foi concluída. Você também pode ter um programa agendado para executar regularmente que, às vezes, precise entrar em contato com você, por exemplo, um programa que verifique a previsão do tempo e envie uma mensagem de texto para lembrá-lo de pegar um guarda-chuva.

Como um exemplo simples, apresentamos a seguir um pequeno programa Python com uma função textmyself() que envia uma mensagem recebida como argumento na forma de string. Abra uma nova janela no editor de arquivo e digite o código a seguir, substituindo o SID da conta, o token de autenticação e os números de telefone pelas suas próprias informações. Salve-o como *textMyself.py*.

```
#! python3
# textMyself.py – Define a função textmyself() que envia uma mensagem de texto
# passada a ela como uma string.

# Valores predefinidos::
accountSID = 'ACxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx'
authToken  = 'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx'
myNumber = '+15559998888'
twilioNumber = '+15552225678'

from twilio.rest import TwilioRestClient

u def textmyself(message):
v     twilioCli = TwilioRestClient(accountSID, authToken)
w     twilioCli.messages.create(body=message, from_=twilioNumber, to=myNumber)
```

Esse programa armazena um SID de conta, o token de autenticação, o número que enviará a mensagem e o número que a receberá. Em seguida, ele define textmyself() para que receba um argumento u, cria um objeto TwilioRestClient e chama create() com a mensagem recebida w.

Se quiser disponibilizar a função textmyself() a outros programas, basta colocar o arquivo *textMyself.py* na mesma pasta em que estiver o executável do Python (*C:\Python34* no Windows, */usr/local/lib/python3.4* no OS X e */usr/bin/python3* no Linux). Agora você poderá usar essa função em outros programas. Sempre que quiser que um de seus programas envie uma mensagem de texto para você, basta acrescentar o seguinte:

```
import textmyself
textmyself.textmyself('The boring task is finished.')
```

Será necessário inscrever-se junto ao Twilio e criar o código para o envio de mensagens de texto somente uma vez. Depois disso, bastam duas linhas de código para que seus outros programas possam enviar uma mensagem de texto.

Resumo

Nós nos comunicamos uns com os outros pela Internet e pelas redes de telefones celulares por meio de dezenas de modos diferentes, porém o email e as mensagens de texto são predominantes. Seus programas podem se comunicar por meio desses canais, o que lhes permite ter novos recursos eficazes para notificação. Você pode até mesmo criar programas que executem em computadores diferentes e que se comuniquem diretamente uns com os outros por email, com um programa enviando emails com SMTP e o outro recebendo-os com o IMAP.

O smtplib do Python disponibiliza funções para usar o SMTP e enviar emails por meio do servidor SMTP de seu provedor de emails. De modo semelhante, os módulos imapclient e pyzmail de terceiros permitem acessar servidores IMAP e obter os emails enviados a você. Embora o IMAP seja um pouco mais complexo que o SMTP, ele também é bastante eficaz e permite pesquisar emails em particular, fazer seu download e efetuar parse para extrair o assunto e o corpo dos emails na forma de valores em string.

Enviar mensagens de texto é um processo um pouco diferente de enviar emails, pois, de modo diferente dos emails, mais do que uma simples conexão com a Internet será necessária para enviar mensagens SMS. Felizmente, serviços como o Twilio disponibilizam módulos que permitem

enviar mensagens de texto a partir de seus programas. Após passar pelo processo inicial de configuração, você poderá enviar mensagens de texto com apenas duas linhas de código.

Com esses módulos em seu conjunto de habilidades, você poderá programar condições específicas de acordo com as quais seus programas deverão enviar notificações ou lembretes. A partir de agora, seus programas terão um alcance muito maior, indo além do computador em que estiverem executando!

Exercícios práticos

1. Qual é o protocolo para enviar emails? E para verificar e receber emails?
2. Quais são as quatro funções/métodos de smtplib que devem ser chamados para fazer login em um servidor SMTP?
3. Quais são as duas funções/métodos de imapclient que devem ser chamados para fazer login em um servidor IMAP?
4. Que tipo de argumento deve ser passado para imapObj.search()?
5. O que deverá ser feito se o seu código obtiver uma mensagem de erro contendo got more than 10000 bytes (mais de 10.000 bytes obtidos)?
6. O módulo imapclient cuida da conexão com um servidor IMAP e permite encontrar emails. Qual é o módulo que cuida da leitura dos emails obtidos por imapclient?
7. Quais são as três informações do Twilio necessárias antes de podermos enviar mensagens de texto?

Projetos práticos

Para exercitar, escreva programas que façam as tarefas a seguir.

Programa para enviar emails com atribuições de tarefas aleatórias

Crie um programa que utilize uma lista de endereços de email e uma lista de tarefas a serem feitas e atribua essas tarefas aleatoriamente às pessoas. Envie

as tarefas atribuídas a cada pessoa por email. Se você estiver se sentindo ambicioso, mantenha um registro das tarefas atribuídas anteriormente a cada pessoa de modo que você possa garantir que o programa evite atribuir a mesma tarefa realizada antes à mesma pessoa. Para ter outra funcionalidade possível, agende o programa para que ele seja executado uma vez por semana automaticamente.

Aqui está uma dica: se você passar uma lista à função random.choice(), ela retornará um item selecionado aleatoriamente nessa lista. Parte de seu código poderá ter o seguinte aspecto:

```
chores = ['dishes', 'bathroom', 'vacuum', 'walk dog']
randomChore = random.choice(chores)
chores.remove(randomChore)    # essa tarefa agora já foi atribuída, portanto remova-a
```

Lembrete para pegar o guarda-chuva

No capítulo 11, vimos como usar o módulo requests para extrair dados de *http://weather.gov/*. Crie um programa que execute imediatamente antes de você acordar pela manhã e verifique se vai chover nesse dia. Em caso afirmativo, faça o programa enviar uma mensagem de texto a você como lembrete para pegar um guarda-chuva antes de sair de casa.

Cancelamento automático de inscrição

Crie um programa que verifique sua conta de email, localize todos os links para cancelamento de inscrição em todos os seus emails e abra-os automaticamente em um navegador. Esse programa deverá fazer login no servidor IMAP de seu provedor de emails e fazer o download de todos os seus emails. Você pode usar o BeautifulSoup (discutido no capítulo 11) para localizar todas as instâncias em que a palavra *unsubscribe* ocorra em uma tag de link HTML.

Depois que tiver uma lista com esses URLs, você poderá usar webbrowser.open() para abrir automaticamente todos esses links em um navegador.

Você ainda deverá percorrer manualmente essas listas e completar qualquer passo adicional para cancelar sua inscrição. Na maioria dos casos, isso

envolverá clicar em um link para confirmar.

No entanto esse script evita que você precise acessar todos os seus emails à procura de links para cancelamento de inscrições. Você poderá disponibilizar esse script aos seus amigos para que eles possam executá-lo em suas contas de email. (Certifique-se de que a senha de seu email não esteja fixa no código-fonte!)

Controlando seu computador por email

Crie um programa que verifique uma conta de email a cada 15 minutos em busca de qualquer instrução enviada por email e execute essas instruções automaticamente. Por exemplo, o BitTorrent é um sistema de downloading peer-to-peer. Ao usar um software gratuito de BitTorrent como o qBittorrent, podemos fazer download de arquivos grandes de mídia em nossos computadores domésticos. Se você enviar um link BitTorrent (totalmente legal, sem ser pirata) por email ao programa, em algum momento, ele verificará seus emails, encontrará essa mensagem, extrairá o link e iniciará o qBittorrent para iniciar o download do arquivo. Dessa maneira, você poderá fazer o seu computador doméstico iniciar downloads enquanto você estiver distante dele e o download (totalmente legal, sem ser pirata) poderá estar concluído quando você voltar para casa.

O capítulo 15 discute o modo de iniciar programas em seu computador usando a função `subprocess.Popen()`. Por exemplo, a chamada a seguir iniciará o programa qBittorrent, juntamente com um arquivo torrent:

```
qbProcess = subprocess.Popen(['C:\\Program Files (x86)\\qBittorrent\\qbittorrent.exe',
'shakespeare_complete_works.torrent'])
```

É claro que você vai querer que o programa garanta que os emails tenham sido enviados por você. Em particular, você poderá exigir que os emails contenham uma senha, pois é muito fácil aos hackers criar um endereço “from” (de) falso nos emails. O programa deve apagar os emails encontrados para que as instruções não sejam repetidas sempre que a conta de email for verificada. Como funcionalidade adicional, faça o programa enviar uma confirmação por email ou com uma mensagem de texto sempre que ele executar um comando. Como você não estará diante do computador que

estará executando o programa, usar funções de logging (veja o capítulo 10) para gravar logs em um arquivo-texto que possa ser verificado caso haja algum erro é uma boa ideia.

O qBittorrent (assim como outros aplicativos para BitTorrent) tem um recurso que lhe permite terminar automaticamente após o download ter sido concluído. O capítulo 15 explica como podemos determinar se uma aplicação iniciada terminou usando o método wait() de objetos Popen. A chamada ao método wait() ficará bloqueada até o qBittorrent terminar; então seu programa poderá enviar um email ou uma mensagem de texto a você com uma notificação para informar que o download foi concluído.

Há muitas funcionalidades possíveis que poderão ser acrescentadas a esse projeto. Se não souber o que fazer, você poderá fazer o download de um exemplo de implementação desse programa a partir de *http://nostarch.com/automatestuff/*.

CAPÍTULO 17

MANIPULANDO IMAGENS

Se você tem uma câmera digital ou se apenas faz upload de fotos de seu telefone para o Facebook, é provável que você vá se deparar com arquivos de imagens digitais o tempo todo. Talvez você saiba usar softwares gráficos básicos como o Microsoft Paint ou o Paintbrush ou até mesmo aplicativos mais sofisticados como o Adobe Photoshop. Porém, se precisar editar uma grande quantidade de imagens, fazê-lo manualmente poderá ser uma tarefa demorada e maçante.

Use o Python. O Pillow é um módulo Python de terceiros para interagir com arquivos de imagem. O módulo contém diversas funções que facilitam cortar, redimensionar e editar o conteúdo de uma imagem. Com a capacidade de manipular imagens do mesmo modo que você faria com um software como o Microsoft Paint ou o Adobe Photoshop, o Python poderá editar facilmente centenas ou milhares de imagens de maneira automática.

Básico sobre imagens no computador

Para manipular uma imagem, é preciso entender o básico sobre como os computadores lidam com cores e coordenadas de imagens e como podemos trabalhar com cores e coordenadas no Pillow. Porém, antes de prosseguir, instale o módulo pillow. Veja o apêndice A para obter ajuda com a instalação de módulos de terceiros.

Cores e valores RGBA

Geralmente, os programas de computador representam uma cor em uma imagem como um *valor RGBA*. Um valor RGBA é um grupo de números que especifica a quantidade de vermelho (red), verde (green), azul (blue) e *alpha* (transparência) em uma cor. Os valores de cada um desses componentes são um inteiro de 0 (nada) a 255 (o máximo). Esses valores de RGBA são

atribuídos a *pixels* individuais; um pixel é o menor ponto de uma única cor que a tela do computador é capaz de mostrar (como você pode imaginar, há milhões de pixels em uma tela). A configuração RBG de um pixel informa exatamente o tom da cor a ser exibida. As imagens também têm um valor alpha para criar valores RGBA. Se uma imagem for exibida na tela sobre uma imagem de fundo ou sobre o papel de parede do desktop, o valor alpha determinará quanto do plano de fundo poderá ser visto "através" do pixel da imagem.

No Pillow, os valores RGBA são representados por uma tupla de quatro valores inteiros. Por exemplo, a cor vermelha é representada por (255, 0, 0, 255). Essa cor tem a quantidade máxima de vermelho, nada de verde ou de azul e o valor máximo de alpha, o que quer dizer que é totalmente opaca. O verde é representado por (0, 255, 0, 255) e o azul, por (0, 0, 255, 255). O branco, que é a combinação de todas as cores, é (255, 255, 255, 255), enquanto o preto, que representa a ausência total de cor, é (0, 0, 0, 255).

Se uma cor tiver um valor alpha igual a 0, ela será invisível e seus valores RGB não serão realmente importantes. Afinal de contas, vermelho invisível será o mesmo que preto invisível.

O Pillow utiliza os nomes padronizados de cores usados pelo HTML. A tabela 17.1 apresenta uma seleção dos nomes padronizados de cores e seus valores.

O Pillow disponibiliza a função ImageColor.getcolor() para que não seja necessário memorizar os valores RGBA das cores que você quiser usar. Essa função aceita uma string com o nome de uma cor como seu primeiro argumento e a string 'RGBA' como segundo argumento e retorna uma tupla RGBA.

Tabela 17.1 – Nomes padronizados de cores e seus valores RGBA

Nome	Valor RGBA	Nome	Valor RGBA
White (branco)	(255, 255, 255, 255)	Red (vermelho)	(255, 0, 0, 255)
Green (verde)	(0, 128, 0, 255)	Blue (azul)	(0, 0, 255, 255)
Gray (cinza)	(128, 128, 128, 255)	Yellow (amarelo)	(255, 255, 0, 255)
Black (preto)	(0, 0, 0, 255)	Purple (roxo)	(128, 0, 128, 255)

CORES CMYK E RGB

No ensino fundamental, você aprendeu que misturar as tintas vermelha, amarela e azul pode gerar outras cores; por exemplo, podemos misturar azul e amarelo para obter uma tinta verde. Isso é conhecido como *modelo subtrativo de cores* e aplica-se a corantes, tintas e pigmentos. É por isso que as impressoras coloridas usam cartuchos de tinta *CMYK*: as tintas *Cyan* (azul), *Magenta* (vermelho), *Yellow* (amarelo) e *black* (preto) podem ser misturadas para formar qualquer cor.

Entretanto a física das luzes utiliza o que chamamos de *modelo aditivo de cores*. Ao utilizar luzes (por exemplo, a luz emitida pela tela de seu computador), as luzes vermelha, verde e azul podem ser combinadas para formar qualquer outra cor. É por isso que os valores *RGB* representam as cores em seus programas de computador.

Para ver o funcionamento de ImageColor.getcolor(), digite o seguinte no shell interativo:

```
u >>> from PIL import ImageColor
v >>> ImageColor.getcolor('red', 'RGBA')
  (255, 0, 0, 255)
w >>> ImageColor.getcolor('RED', 'RGBA')
  (255, 0, 0, 255)
  >>> ImageColor.getcolor('Black', 'RGBA')
  (0, 0, 0, 255)
  >>> ImageColor.getcolor('chocolate', 'RGBA')
  (210, 105, 30, 255)
  >>> ImageColor.getcolor('CornflowerBlue', 'RGBA')
  (100, 149, 237, 255)
```

Inicialmente, é preciso importar o módulo ImageColor de PIL u (e não de Pillow; você verá por quê em breve). A string com o nome da cor passada para ImageColor.getcolor() não diferencia letras maiúsculas de minúsculas, portanto passar 'red' v e 'RED' w resultará na mesma tupla RGBA. Você também pode passar nomes mais incomuns de cores como 'chocolate' e 'CornflowerBlue'.

O Pillow suporta uma quantidade enorme de nomes de cores, de 'aliceblue' a 'whitesmoke'. A lista completa com mais de cem nomes-padrões de cores está disponível nos recursos em *http://nostarch.com/automatestuff/*.

Coordenadas e tuplas de caixa

Os pixels das imagens são acessíveis por meio de coordenadas x e y que especificam, respectivamente, uma posição horizontal e uma posição vertical de um pixel em uma imagem. A *origem* é o pixel no canto superior esquerdo da imagem e é especificada com a notação (0, 0). O primeiro zero representa a coordenada x, que começa em zero na origem e aumenta da esquerda para a direita. O segundo zero representa a coordenada y, que começa em zero na origem e aumenta de cima para baixo na imagem. Vale a pena repetir isso: as coordenadas y aumentam para baixo, que é oposto ao modo como você pode se recordar do uso das coordenadas y nas aulas de matemática. A figura 17.1 mostra como esse sistema de coordenadas funciona.

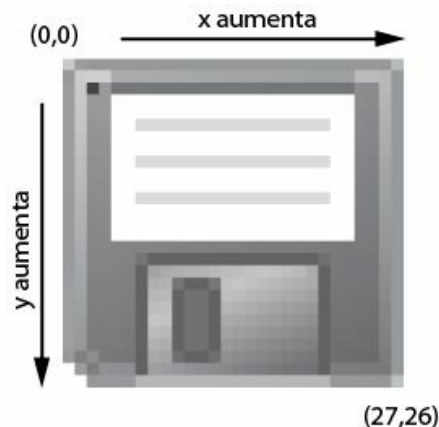



Figura 17.1 – As coordenadas x e y de uma imagem de 27 × 26 de uma espécie antiga de dispositivo de armazenamento de dados.

Muitas funções e vários métodos do Pillow aceitam um argumento que representa uma *tupla de caixa* (box tuple). Isso que dizer que o Pillow espera uma tupla com quatro coordenadas inteiras que represente uma região retangular em uma imagem. Os quatro inteiros, em sequência, são:

- *Esquerda (left)* – A coordenada x da borda esquerda da caixa.
- *Parte superior (top)* – A coordenada y da borda superior da caixa.
- *Direita (right)* – A coordenada x de um pixel à direita da borda direita da caixa; esse inteiro deve ser maior que o inteiro que representa a esquerda.
- *Parte inferior (bottom)* – A coordenada y de um pixel abaixo da borda inferior da caixa; esse inteiro deve ser maior que o inteiro que representa a parte superior.

Observe que a caixa inclui as coordenadas esquerda e superior e avança até – mas não inclui – as coordenadas direita e inferior. Por exemplo, a tupla de caixa (3, 1, 9, 6) representa todos os pixels da caixa preta da figura 17.2.

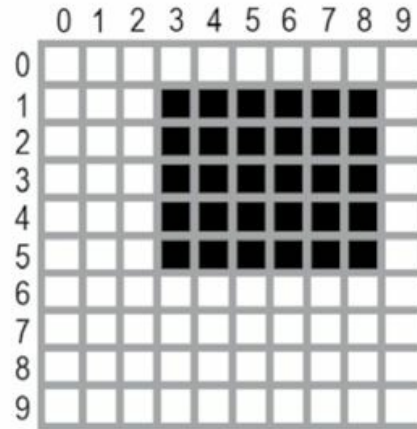



Figura 17.2 – A área representada pela tupla de caixa (3, 1, 9, 6).

Manipulando imagens com o Pillow

Agora que sabemos como as cores e as coordenadas funcionam no Pillow, vamos usá-lo para manipular uma imagem. A figura 17.3 contém a imagem que será usada em todos os exemplos no shell interativo neste capítulo. Seu download pode ser feito a partir de *http://nostarch.com/automatestuff/*.

Depois que o arquivo de imagem *Zophie.png* estiver em seu diretório de trabalho atual, você estará pronto para carregar a imagem de Zophie no Python da seguinte maneira:

```
>>> from PIL import Image
>>> catIm = Image.open('zophie.png')
```

Figura 17.3 – Minha gata Zophie. A câmera lhe acrescenta cinco quilos (o que é muito para um gato).

Para carregar a imagem, importe o módulo Image do Pillow e chame Image.open() passando-lhe o nome do arquivo de imagem. Então você poderá armazenar a imagem carregada em uma variável como catIm. O nome do módulo do Pillow é PIL para que seja compatível com versões mais antigas de um módulo chamado Python Imaging Library, motivo pelo qual você deve executar from PIL import Image em vez de from Pillow import Image. Por causa da maneira pela qual os criadores do Pillow definiram o módulo pillow, você deve usar a forma from PIL import Image da instrução import em vez de simplesmente utilizar import PIL.

Se o arquivo de imagem não estiver no diretório de trabalho atual, mude o diretório de trabalho para a pasta que contém esse arquivo chamando a função os.chdir().

```
>>> import os
>>> os.chdir('C:\\folder_with_image_file')
```

A função Image.open() retorna um valor cujo tipo de dado é um objeto Image, que é como o Pillow representa uma imagem na forma de um valor Python. Você pode carregar um objeto Image a partir de um arquivo de imagem (em qualquer formato) ao passar uma string com o nome do arquivo à função

Image.open(). Qualquer alteração feita em um objeto Image poderá ser salvada em um arquivo de imagem (também em qualquer formato) com o método save(). Todas as rotações, os redimensionamentos, os recortes, os desenhos e outras manipulações de imagens serão feitos por meio de chamadas de métodos nesse objeto Image.

Para deixar os exemplos deste capítulo mais compactos, vou supor que você importou o módulo Image do Pillow e que tem a imagem de Zophie armazenada em uma variável chamada catIm. Certifique-se de que o arquivo *zophie.png* esteja no diretório de trabalho atual para que a função Image.open() possa encontrá-lo. Caso contrário, você deverá especificar também o path absoluto completo no argumento de string para Image.open().

Trabalhando com o tipo de dado Image

Um objeto Image tem diversos atributos úteis que fornecem informações básicas sobre o arquivo de imagem a partir do qual esse objeto foi carregado: sua largura e sua altura, o nome do arquivo e o formato da imagem (por exemplo, JPEG, GIF ou PNG).

Por exemplo, digite o seguinte no shell interativo:

```
  >>> from PIL import Image
  >>> catIm = Image.open('zophie.png')
  >>> catIm.size
u (816, 1088)
v >>> width, height = catIm.size
w >>> width
  816
x >>> height
  1088
  >>> catIm.filename
  'zophie.png'
  >>> catIm.format
  'PNG'
  >>> catIm.format_description
  'Portable network graphics'
y >>> catIm.save('zophie.jpg')
```

Após ter criado um objeto Image a partir de *Zophie.png* e de ter armazenado

esse objeto em catIm, podemos ver que o atributo size do objeto contém uma tupla com a largura e a altura da imagem em pixels u. Podemos atribuir os valores da tupla às variáveis width e height v para acessar sua largura w e a sua altura x individualmente. O atributo filename descreve o nome do arquivo original. Os atributos format e format_description são strings que descrevem o formato da imagem no arquivo original (format_description é um pouco mais extenso).

Por fim, chamar o método save() e passar-lhe 'zophie.jpg' salva uma nova imagem com o nome de arquivo *zophie.jpg* em seu disco rígido y. O Pillow percebe que a extensão do arquivo é *.jpg* e salva automaticamente a imagem usando o formato de imagem JPEG. Agora você deverá ter duas imagens, *zophie.png* e *zophie.jpg*, em seu disco rígido. Embora esses arquivos sejam baseados na mesma imagem, eles não são idênticos por causa de seus formatos diferentes.

O Pillow também disponibiliza a função Image.new(), que retorna um objeto Image – de modo muito semelhante a Image.open(), exceto pelo fato de a imagem representada pelo objeto de Image.new() estar em branco. Os argumentos de Image.new() são:

- A string 'RGBA', que define o modo de cor para RGBA. (Há outros modos, mas eles não serão descritos neste livro.)
- O tamanho na forma de uma tupla com dois inteiros, referente à largura e à altura da nova imagem.
- A cor de fundo inicial da imagem na forma de uma tupla de quatro inteiros, referente a um valor RGBA. O valor de retorno da função ImageColor.getcolor() pode ser usado para esse argumento. De modo alternativo, Image.new() também aceita somente a string com o nome-padrão da cor.

Por exemplo, digite o seguinte no shell interativo:

```
  >>> from PIL import Image
u >>> im = Image.new('RGBA', (100, 200), 'purple')
  >>> im.save('purpleImage.png')
v >>> im2 = Image.new('RGBA', (20, 20))
  >>> im2.save('transparentImage.png')
```

Nesse caso, criamos um objeto Image para uma imagem com 100 pixels de largura e 200 pixels de altura, com uma cor de fundo roxa (purple) u. Essa imagem é então salvada no arquivo *purpleImage.png*. Chamamos Image.new() novamente para criar outro objeto Image, desta vez, passando (20, 20) para as dimensões e nenhum dado para a cor de fundo v. O preto invisível, ou seja, (0, 0, 0, 0), é a cor default usada caso nenhum argumento referente à cor seja especificado, portanto a segunda imagem tem um plano de fundo transparente; salvamos esse quadrado transparente de 20 × 20 em *transparentImage.png*.

Recortando imagens

Recortar uma imagem (cropping) quer dizer selecionar uma região retangular em uma imagem e remover tudo que estiver fora do retângulo. O método crop() de um objeto Image aceita uma tupla que representa uma caixa e retorna um objeto Image que representa a imagem recortada. O recorte não ocorre in place (no local) – ou seja, o objeto Image original permanece inalterado e o método crop() retorna um novo objeto Image. Lembre-se de que uma tupla de caixa – nesse caso, a seção recortada – inclui a coluna esquerda e a linha superior de pixels e avança até a coluna direita e a linha inferior de pixels, porém *sem* incluí-las.

Digite o seguinte no shell interativo:

```
>>> croppedIm = catIm.crop((335, 345, 565, 560))
>>> croppedIm.save('cropped.png')
```

Essas instruções criam um novo objeto Image com a imagem recortada, armazena o objeto em croppedIm e chama save() em croppedIm para salvar a imagem recortada em *cropped.png*. O novo arquivo *cropped.png* será criado a partir da imagem original, como mostra a figura 17.4.

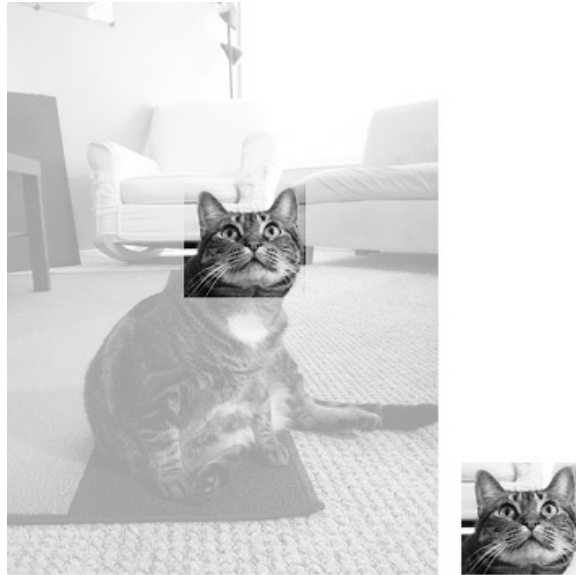



Figura 17.4 – A nova imagem será simplesmente a seção recortada da imagem original.

Copiando e colando imagens sobre outras imagens

O método copy() retorna um novo objeto Image com a mesma imagem do objeto Image em que esse método for chamado. Isso será conveniente se você precisar fazer alterações em uma imagem, mas quiser também manter a versão original inalterada. Por exemplo, digite o seguinte no shell interativo:

```
>>> catIm = Image.open('zophie.png')
>>> catCopyIm = catIm.copy()
```

As variáveis catIm e catCopyIm contêm dois objetos Image diferentes, mas ambos têm a mesma imagem. Agora que temos um objeto Image armazenado em catCopyIm, podemos modificá-lo como quisermos e salvá-lo em um novo arquivo, deixando *zophie.png* inalterado. Por exemplo, vamos experimentar modificar catCopyIm com o método paste().

O método paste() é chamado em um objeto Image e cola outra imagem sobre ela. Vamos prosseguir com o exemplo no shell e colar uma imagem menor sobre catCopyIm.

```
>>> faceIm = catIm.crop((335, 345, 565, 560))
>>> faceIm.size
(230, 215)
>>> catCopyIm.paste(faceIm, (0, 0))
>>> catCopyIm.paste(faceIm, (400, 500))
```

```
>>> catCopyIm.save('pasted.png')
```

Inicialmente, passamos uma tupla de caixa para crop() com a área retangular em *zophie.png*, que contém a cara de Zophie. Isso cria um objeto Image representando um recorte de 230 × 215 que será armazenado em faceIm. Agora podemos colar faceIm sobre catCopyIm. O método paste() aceita dois argumentos: um objeto Image de "origem" e uma tupla com as coordenadas x e y do local em que o canto superior esquerdo do objeto Image de origem será colado sobre o objeto Image principal. Nesse caso, chamamos paste() duas vezes em catCopyIm passando (0, 0) na primeira vez e (400, 500) na segunda vez. Isso faz faceIm ser colado em catCopyIm duas vezes: uma vez com o canto superior esquerdo de faceIm em (0, 0) em catCopyIm e a outra vez com o canto superior esquerdo de faceIm em (400, 500). Por fim, salvamos o catCopyIm modificado em *pasted.png*. A imagem de *pasted.png* terá a aparência mostrada na figura 17.5.

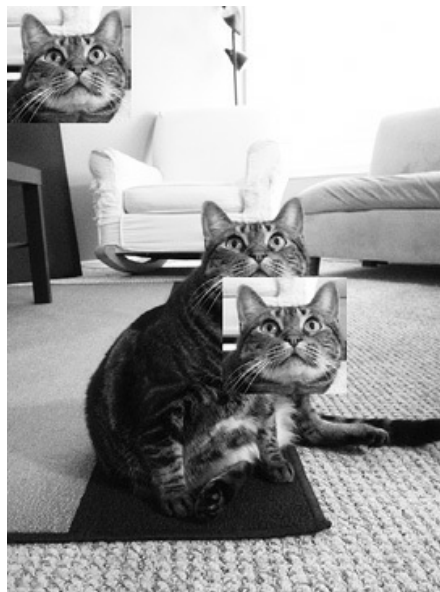



Figura 17.5 – A gata Zophie com sua cara colada duas vezes.

NOTA Apesar de seus nomes, os métodos copy() e paste() do Pillow não usam o clipboard (área de transferência) de seu computador.

Observe que o método paste() modifica seu objeto Image *in place* (no local); ele não retorna um objeto Image com a imagem colada. Se quiser chamar paste(), mas quiser também preservar uma versão inalterada da imagem original, será necessário inicialmente copiar a imagem e, em seguida, chamar

paste() nessa cópia.

Suponha que você queira colocar a cara de Zophie lado a lado em toda a imagem, conforme mostrado na figura 17.6. Esse efeito pode ser obtido com apenas dois loops for. Prossiga com o exemplo no shell interativo digitando o seguinte:

```
  >>> catImWidth, catImHeight = catIm.size
  >>> faceImWidth, faceImHeight = faceIm.size
u >>> catCopyTwo = catIm.copy()
v >>> for left in range(0, catImWidth, faceImWidth):
w         for top in range(0, catImHeight, faceImHeight):
              print(left, top)
              catCopyTwo.paste(faceIm, (left, top))
  0 0
  0 215
  0 430
  0 645
  0 860
  0 1075
  230 0
  230 215
  --trecho removido--
  690 860
  690 1075
  >>> catCopyTwo.save('tiled.png')
```

Nesse caso, armazenamos a largura e a altura de catIm em catImWidth e em catImHeight. Em u criamos uma cópia de catIm e a armazenamos em catCopyTwo. Agora que temos uma cópia sobre a qual podemos colar uma imagem, iniciamos o loop para colar faceIm em catCopyTwo. A variável left do loop for externo começa em 0 e é incrementada de faceImWidth(230) v. A variável top do loop for interno começa em 0 e é incrementada de faceImHeight(215) w. Esses loops for aninhados geram valores para left e top para que uma grade de imagens faceIm seja colada sobre o objeto Image em catCopyTwo, como mostra a figura 17.6. Para ver nossos loops aninhados em funcionamento, exibimos left e top. Após a colagem estar concluída, salvamos o catCopyTwo modificado em *tiled.png*.

Figura 17.6 – Loops for *aninhados usados com* paste() *para duplicar a cara da gata.*

COLANDO PIXELS TRANSPARENTES

Normalmente, os pixels transparentes são colados como pixels brancos. Se a imagem que você quiser colar tiver pixels transparentes, passe o objeto Image como o terceiro argumento para que um retângulo sólido não seja colado. O terceiro argumento é o objeto Image que serve como "máscara". Uma máscara é um objeto Image em que o valor alpha é significativo, porém os valores para vermelho, verde e azul são ignorados. A máscara informa à função paste() quais pixels devem ser copiados e quais devem ser deixados como transparentes. Um uso mais avançado de máscaras está além do escopo deste livro, porém, se quiser colar uma imagem que tenha pixels transparentes, passe o objeto Image novamente como o terceiro argumento.

Redimensionando uma imagem

O método resize() é chamado em um objeto Image e retorna um novo objeto Image com a largura e a altura especificadas. Ele aceita uma tupla composta de dois inteiros como argumento, que representa a nova largura e a nova altura da imagem retornada. Digite o seguinte no shell interativo:

```
u >>> width, height = catIm.size
v >>> quartersizedIm = catIm.resize((int(width / 2), int(height / 2)))
```

```
>>> quartersizedIm.save('quartersized.png')
w >>> svelteIm = catIm.resize((width, height + 300))
>>> svelteIm.save('svelte.png')
```

Nesse caso, atribuímos os dois valores da tupla catIm.size às variáveis width e height u. Usar width e height no lugar de catIm.size[0] e de catIm.size[1] torna o restante do código mais legível.

Na primeira chamada a resize(), passamos int(width / 2) como a nova largura e int(height / 2) como a nova altura v; desse modo, o objeto Image retornado por resize() terá metade da altura e da largura da imagem original, ou seja, um quarto do tamanho dessa imagem. O método resize() aceita somente inteiros em seu argumento de tupla, motivo pelo qual é preciso inserir as duas divisões por 2 em uma chamada a int().

Esse redimensionamento mantém as mesmas proporções para a largura e para a altura. Porém a nova largura e a nova altura passadas para resize() não precisam ser proporcionais à imagem original. A variável svelteIm contém um objeto Image que tem a largura original, porém a altura é 300 pixels maior w, dando a Zophie uma aparência mais alongada.

Observe que o método resize() não altera o objeto Image in place, mas retorna um novo objeto Image.

Fazendo rotações e invertendo as imagens

As imagens podem sofrer rotações com o método rotate(), que retorna um novo objeto Image com a imagem girada e deixa o objeto Image original inalterado. O argumento de rotate() é um único inteiro ou número de ponto flutuante que representa o número de graus segundo o qual a imagem será girada em sentido anti-horário. Digite o seguinte no shell interativo:

```
>>> catIm.rotate(90).save('rotated90.png')
>>> catIm.rotate(180).save('rotated180.png')
>>> catIm.rotate(270).save('rotated270.png')
```

Observe como podemos *encadear* as chamadas de métodos ao chamar save() diretamente no objeto Image retornado por rotate(). A primeira chamada a rotate() e a save() criam um novo objeto Image que representa a imagem girada em 90 graus no sentido anti-horário e salva essa imagem girada em

rotated90.png. A segunda e a terceira chamadas fazem o mesmo, porém usam 180 graus e 270 graus. Os resultados têm a aparência mostrada na figura 17.7.

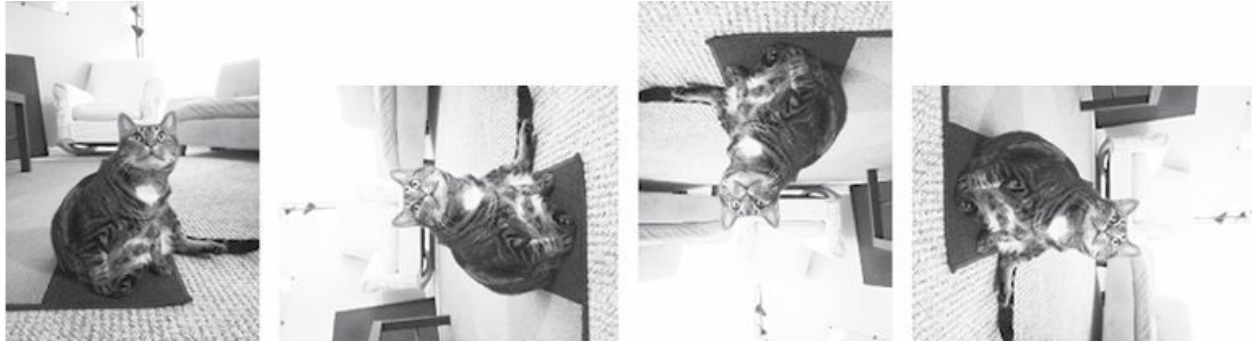



Figura 17.7 – A imagem original (à esquerda) e a imagem girada no sentido anti-horário em 90 graus, 180 graus e 270 graus.

Observe que a largura e a altura da imagem mudam quando a imagem é girada em 90 graus ou em 270 graus. Se você fizer a rotação de uma imagem de acordo com outro valor, as dimensões originais da imagem serão preservadas. No Windows, um fundo preto é usado para preencher qualquer lacuna criada pela rotação, como mostra a figura 17.8. No OS X, pixels transparentes são usados nas lacunas.

O método rotate() tem um argumento nomeado expand opcional que pode ser definido com True para aumentar as dimensões da imagem de modo que ela se enquadre totalmente na nova imagem girada. Por exemplo, digite o seguinte no shell interativo:

```
>>> catIm.rotate(6).save('rotated6.png')
>>> catIm.rotate(6, expand=True).save('rotated6_expanded.png')
```

A primeira chamada faz a rotação da imagem em 6 graus e a salva em *rotate6.png* (veja a imagem na figura 17.8 à esquerda). A segunda chamada faz a rotação da imagem em 6 graus com expand definido com True e a salva em *rotate6_expanded.png* (veja a imagem na figura 17.8 à direita).

Também podemos obter uma imagem "espelhada" usando o método transpose(). Passe Image.FLIP_LEFT_RIGHT ou Image.FLIP_TOP_BOTTOM ao método transpose(). Digite o seguinte no shell interativo:

```
>>> catIm.transpose(Image.FLIP_LEFT_RIGHT).save('horizontal_flip.png')
```

```
>>> catIm.transpose(Image.FLIP_TOP_BOTTOM).save('vertical_flip.png')
```

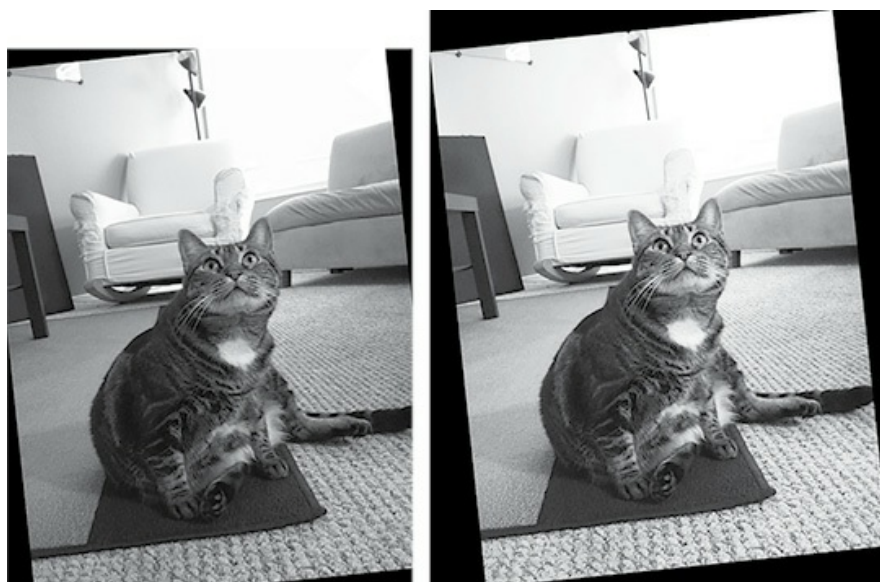



Figura 17.8 – A imagem girada normalmente em 6 graus (à esquerda) e com expand=True *(à direita).*

Assim como rotate(), transpose() cria um novo objeto Image. Nesse caso, passamos Image.FLIP_LEFT_RIGHT para inverter a imagem horizontalmente e, em seguida, salvamos o resultado em *horizontal_flip.png*. Para inverter a imagem verticalmente, passamos Image.FLIP_TOP_BOTTOM e a salvamos em *vertical_flip.png*. Os resultados têm a aparência mostrada na figura 17.9.

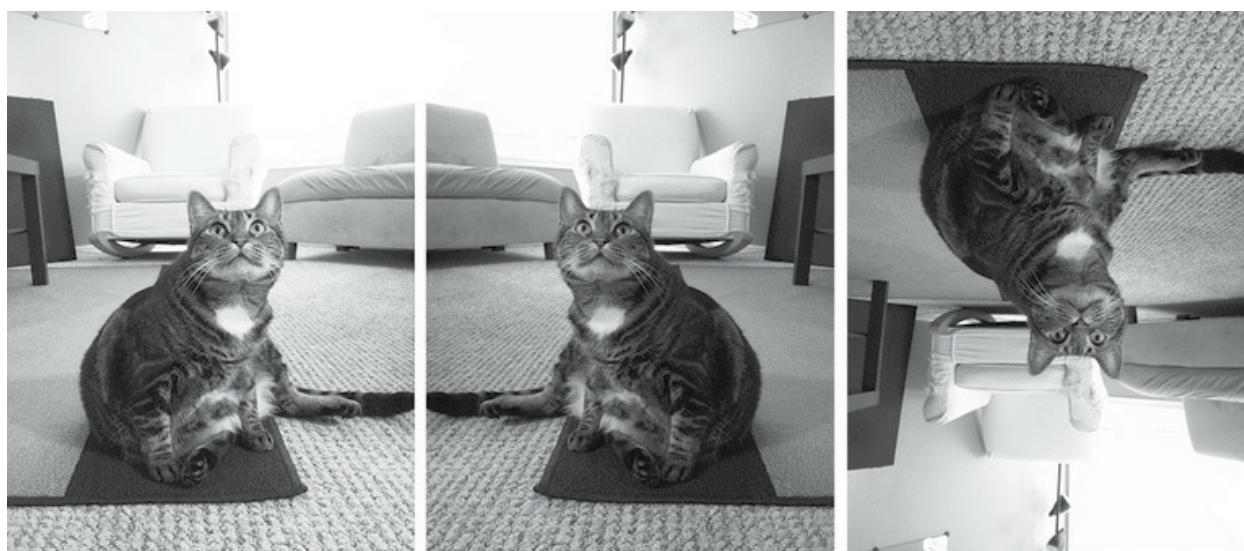



Figure 17.9 – A imagem original (à esquerda), invertida horizontalmente (no centro) e invertida verticalmente (à direita).

Alterando pixels individuais

A cor de um pixel individual pode ser obtida ou definida com os métodos getpixel() e putpixel(). Ambos os métodos aceitam uma tupla que representa as coordenadas x e y do pixel. O método putpixel() também aceita um argumento adicional do tipo tupla para a cor do pixel. Esse argumento de cor corresponde a uma tupla RGBA com quatro inteiros ou a uma tupla RGB com três inteiros. Digite o seguinte no shell interativo:

```
u >>> im = Image.new('RGBA', (100, 100))
v >>> im.getpixel((0, 0))
  (0, 0, 0, 0)
w >>> for x in range(100):
        for y in range(50):
x           im.putpixel((x, y), (210, 210, 210))

  >>> from PIL import ImageColor
y >>> for x in range(100):
        for y in range(50, 100):
z           im.putpixel((x, y), ImageColor.getcolor('darkgray', 'RGBA'))
  >>> im.getpixel((0, 0))
  (210, 210, 210, 255)
  >>> im.getpixel((0, 50))
  (169, 169, 169, 255)
  >>> im.save('putPixel.png')
```

Em u criamos uma nova imagem que corresponde a um quadrado transparente de 100 × 100. A chamada a getpixel() em algumas coordenadas dessa imagem retorna (0, 0, 0, 0) porque a imagem é transparente v. Para colorir os pixels dessa imagem, podemos usar loops for aninhados e percorrer todos os pixels da metade superior da imagem w, colorindo cada pixel com putpixel() x. Nesse caso, passamos a tupla RGB (210, 210, 210), que corresponde a um cinza-claro, a putpixel().

Suponha que você queira colorir a metade inferior da imagem com cinza-escuro, porém não saiba qual é a tupla RGB para cinza-escuro. O método putpixel() não aceita um nome de cor padrão como 'darkgray', portanto será necessário usar ImageColor.getcolor() para obter uma tupla de cor para 'darkgray'. Percorra os pixels da metade inferior da imagem y e passe o valor de retorno

de ImageColor.getcolor() a putpixel() z; agora você deverá ter uma imagem que seja cinza-claro na metade superior e cinza-escuro na metade inferior, conforme mostrado na figura 17.10. Você pode chamar getpixel() em algumas coordenadas para confirmar se qualquer pixel especificado tem a cor esperada. Por fim, salve a imagem em *putPixel.png*.

É claro que desenhar um pixel de cada vez em uma imagem não é muito conveniente. Se for necessário desenhar formas, utilize as funções de ImageDraw explicadas mais adiante neste capítulo.

Figura 17.10 – A imagem em putPixel.png.

Projeto: Adicionando um logo

Suponha que você tenha a tarefa maçante de redimensionar milhares de imagens e adicionar um pequeno logo como marca-d'água no canto de cada uma delas. Fazer isso com um programa gráfico básico como o Paintbrush ou o Paint demoraria muito. Uma aplicação gráfica mais sofisticada como o Photoshop pode fazer processamento em batch (lote), porém esse software custa centenas de dólares. Vamos criar um script para fazer isso.

Suponha que a figura 17.11 corresponda ao logo que você deseja adicionar no canto inferior direito de cada imagem: um ícone de um gato preto com bordas brancas e o restante da imagem transparente.

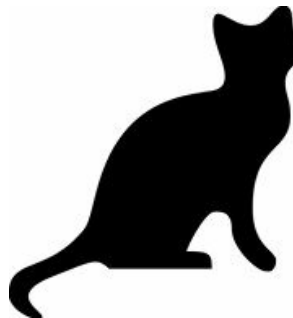



Figura 17.11 – O logo a ser adicionado à imagem.

De modo geral, eis o que o programa deverá fazer:

- Carregar a imagem com o logo.
- Percorrer todos os arquivos *.png* e*.jpg* no diretório de trabalho em um loop.
- Verificar se a imagem tem mais de 300 pixels de largura ou de altura.
- Em caso afirmativo, reduzir a largura ou a altura (o valor maior) para 300 pixels e reduzir a outra dimensão proporcionalmente.
- Colar a imagem do logo no canto.
- Salvar as imagens alteradas em outra pasta.

Isso significa que o código deverá fazer o seguinte:

- Abrir o arquivo *catlogo.png* como um objeto Image.
- Percorrer as strings retornadas por os.listdir('.') em um loop.
- Obter a largura e a altura da imagem a partir do atributo size.
- Calcular a nova largura e a nova altura da imagem redimensionada.
- Chamar o método resize() para redimensionar a imagem.
- Chamar o método paste() para colar o logo.
- Chamar o método save() para salvar as alterações usando o nome original do arquivo.

Passo 1: Abrir a imagem com o logo

Para esse projeto, abra uma nova janela no editor de arquivo, digite o código a seguir e salve-o como *resizeAndAddLogo.py*:

```
   #! python3
   # resizeAndAddLogo.py – Redimensiona todas as imagens do diretório de trabalho atual
   para
   # que caibam em um quadrado de 300x300 e acrescenta catlogo.png no canto inferior
   direito.

   import os
   from PIL import Image

u SQUARE_FIT_SIZE = 300
v LOGO_FILENAME = 'catlogo.png'

w logoIm = Image.open(LOGO_FILENAME)
```

```
x logoWidth, logoHeight = logoIm.size

  # TODO: Percorre todos arquivos do diretório de trabalho em um loop.

  # TODO: Verifica se a imagem deve ser redimensionada.

  # TODO: Calcula a nova largura e a nova altura para o redimensionamento.

  # TODO: Redimensiona a imagem.

  # TODO: Adiciona o logo.

  # TODO: Salva as alterações.
```

Ao configurar as constantes SQUARE_FIT_SIZE u e LOGO_FILENAME v no início, facilitamos futuras alterações no programa. Suponha que o logo que você está adicionando não seja o ícone com o gato ou que você esteja reduzindo a dimensão maior das imagens de saída para um valor diferente de 300 pixels. Com essas constantes no início do programa, você poderá simplesmente abrir o código, alterar esses valores uma vez e pronto. (Ou você pode fazer a implementação de modo que os valores dessas constantes sejam obtidos a partir de argumentos da linha de comando.) Sem essas constantes, seria necessário pesquisar o código em busca de todas as ocorrências de 300 e de 'catlogo.png' e substituí-las pelos valores a serem usados em seu novo projeto. Em suma, usar constantes deixa seu programa mais genérico.

O objeto Image com o logo é retornado por Image.open() w. Por questões de legibilidade, logoWidth e logoHeight recebem os valores de logoIm.size x.

O restante do programa contém um esqueleto com comentários TODO por enquanto.

Passo 2: Percorrer todos os arquivos e abrir as imagens em um loop

Agora devemos encontrar todos os arquivos *.png* e *.jpg* no diretório de trabalho atual. Observe que você não deve adicionar a imagem do logo na própria imagem do logo, portanto o programa deverá ignorar qualquer

imagem cujo nome do arquivo seja igual a LOGO_FILENAME. Adicione o código a seguir:

```
  #! python3
  # resizeAndAddLogo.py – Redimensiona todas as imagens do diretório de trabalho atual
  para
  # que caibam em um quadrado de 300x300 e acrescenta catlogo.png no canto inferior
  direito.

  import os
  from PIL import Image
  --trecho removido--
  os.makedirs('withLogo', exist_ok=True)
  # Percorre todos arquivos do diretório de trabalho em um loop.
u for filename in os.listdir('.'):
v     if not (filename.endswith('.png') or filename.endswith('.jpg')) \
         or filename == LOGO_FILENAME:
w        continue    # ignora os arquivos que não contenham imagens e o próprio arquivo de
logo

x     im = Image.open(filename)
      width, height = im.size
  --trecho removido--
```

Inicialmente, a chamada a os.makedirs() cria uma pasta *withLogo* para armazenar as imagens finalizadas com os logos em vez de sobrescrever os arquivos com as imagens originais. O argumento nomeado exist_ok=True evita que os.makedirs() gere uma exceção caso *withLogo* já exista. Enquanto estiver percorrendo todos os arquivos do diretório de trabalho em um loop com os.listdir('.') u, a instrução if longa v verificará se o nome de cada arquivo termina com *.png* ou com *.jpg*. Em caso afirmativo – ou se for o próprio arquivo com a imagem do logo –, o loop deverá ignorar esse arquivo e usar continue w para acessar o próximo arquivo. Se filename terminar com '.png' ou com '.jpg' (e não for o arquivo de logo), abra-o como um objeto Image x e defina width e height.

Passo 3: Redimensionar as imagens

O programa deve redimensionar a imagem somente se a largura ou a altura

for maior que SQUARE_FIT_SIZE (300 pixels, nesse caso), portanto coloque todo o código de redimensionamento em uma instrução if que verifique as variáveis width e height. Acrescente o código a seguir em seu programa:

```
#! python3
# resizeAndAddLogo.py – Redimensiona todas as imagens do diretório de trabalho atual
para
# que caibam em um quadrado de 300x300 e acrescenta catlogo.png no canto inferior
direito.

import os
from PIL import Image
--trecho removido--
    # Verifica se a imagem deve ser redimensionada.
    if width > SQUARE_FIT_SIZE and height > SQUARE_FIT_SIZE:
        # Calcula a nova largura e a nova altura para o redimensionamento.
        if width > height:
u           height = int((SQUARE_FIT_SIZE / width) * height)
            width = SQUARE_FIT_SIZE
        else:
v           width = int((SQUARE_FIT_SIZE / height) * width)
            height = SQUARE_FIT_SIZE

        # Redimensiona a imagem
        print('Resizing %s...' % (filename))
w       im = im.resize((width, height))
--trecho removido--
```

Se a imagem precisar ser redimensionada, você deverá descobrir se ela é uma imagem larga ou alta. Se width for maior do que height, a altura deverá ser reduzida na mesma proporção que a largura o será u. Essa proporção corresponde ao valor SQUARE_FIT_SIZE dividido pela largura atual. O novo valor de height corresponde a essa proporção multiplicada pelo valor atual de height. Como o operador de divisão retorna um valor de ponto flutuante e resize() exige que as dimensões sejam valores inteiros, lembre-se de converter o resultado para um inteiro usando a função int(). Por fim, o novo valor de width será simplesmente definido com SQUARE_FIT_SIZE.

Se height for maior ou igual a width (ambos os casos são tratados na cláusula else), o mesmo cálculo será feito, exceto pelo fato de as variáveis height e width

serem trocadas v.

Depois que width e height tiverem as dimensões da nova imagem, passe-as ao método resize() e armazene o objeto Image retornado em im w.

Passo 4: Adicionar o logo e salvar as alterações

Independentemente de a imagem ter sido ou não redimensionada, o logo deverá ser colado no canto inferior direito. O local exato em que o logo deve ser colado depende tanto do tamanho da imagem quanto do tamanho do logo. A figura 17.12 mostra como calcular a posição para colar o logo. A coordenada esquerda do local em que o logo deve ser colado será a largura da imagem menos a largura do logo; a coordenada da parte superior da posição em que o logo deverá ser colado será a altura da imagem menos a altura do logo.

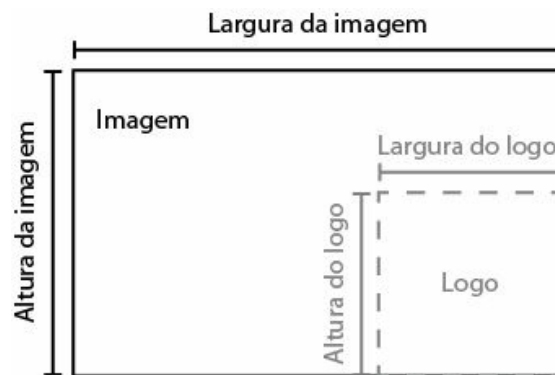



Figura 17.12 – As coordenadas da esquerda e da parte superior para posicionar o logo no canto inferior direito devem ser a largura/altura da imagem menos a largura/altura do logo.

Depois que seu código colar o logo na imagem, o objeto Image modificado deverá ser salvado. Acrescente o seguinte em seu programa:

```
#! python3
# resizeAndAddLogo.py – Redimensiona todas as imagens do diretório de trabalho atual
para
# que caibam em um quadrado de 300x300 e acrescenta catlogo.png no canto inferior
direito.

import os
from PIL import Image
```

```
--trecho removido--

    # Verifica se a imagem deve ser redimensionada.
    --trecho removido--

    # Adiciona o logo.
u   print('Adding logo to %s...' % (filename))
v   im.paste(logoIm, (width - logoWidth, height - logoHeight), logoIm)

    # Salva as alterações.
w   im.save(os.path.join('withLogo', filename))
```

O novo código exibe uma mensagem informando o usuário que o logo está sendo adicionado u, cola logoIm em im nas coordenadas calculadas v e salva as alterações em um arquivo no diretório *withLogo* w. Ao executar esse programa com o arquivo *zophie.png* como a única imagem no diretório de trabalho, a saída será semelhante a:

```
Resizing zophie.png...
Adding logo to zophie.png...
```

A imagem *zophie.png* será alterada para uma imagem de 225 × 300 pixels, semelhante à da figura 17.13. Lembre-se de que o método paste() não colará os pixels transparentes se você não passar logoIm como o terceiro argumento também. Esse programa pode redimensionar automaticamente e inserir logos em centenas de imagens em apenas alguns minutos.

Ideias para programas semelhantes

Ser capaz de compor imagens ou modificar seus tamanhos em batch pode ser útil em diversas aplicações. Você pode criar programas semelhantes para fazer o seguinte:

- Adicionar texto ou o URL de um site às imagens.
- Adicionar timestamps às imagens.
- Copiar ou mover imagens para pastas diferentes de acordo com seus tamanhos.
- Adicionar uma marca-d'água quase transparente em uma imagem para evitar que outras pessoas a copiem.

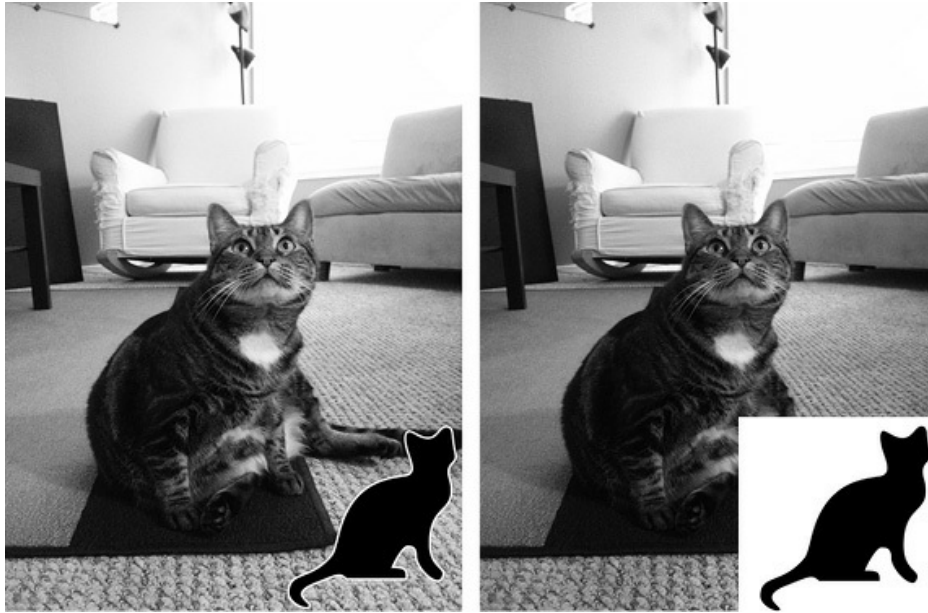



Figura 17.13 – A imagem zophie.png redimensionada e o logo adicionado (à esquerda). Se você se esquecer do terceiro argumento, os pixels transparentes do logo serão copiados como pixels brancos sólidos (à direita).

Desenhando em imagens

Se você precisar desenhar linhas, retângulos, círculos ou outras formas em uma imagem, utilize o módulo ImageDraw do Pillow. Digite o seguinte no shell interativo:

```
>>> from PIL import Image, ImageDraw
>>> im = Image.new('RGBA', (200, 200), 'white')
>>> draw = ImageDraw.Draw(im)
```

Inicialmente, importamos Image e ImageDraw. Em seguida, criamos uma nova imagem – nesse caso, uma imagem branca de 200 × 200 – e armazenamos o objeto Image em im. Passamos o objeto Image para a função ImageDraw.Draw() para que um objeto ImageDraw seja recebido. Esse objeto contém diversos métodos para desenhar formas e texto em um objeto Image. Armazene o objeto ImageDraw em uma variável como draw para que você possa usá-la facilmente no exemplo a seguir.

Desenhando formas

Os métodos de ImageDraw a seguir desenham diversos tipos de formas na

imagem. Os parâmetros fill e outline desses métodos são opcionais e usarão branco como default se não forem especificados.

Pontos

O método point(*xy*, *fill*) desenha pixels individuais. O argumento *xy* representa uma lista de pontos que você quer desenhar. A lista pode ser uma lista de tuplas com coordenadas x e y, por exemplo, [(x, y), (x, y), ...], ou uma lista de coordenadas x e y sem tuplas, como [x1, y1, x2, y2, ...]. O argumento *fill* corresponde à cor dos pontos e pode ser uma tupla RGBA ou uma string com um nome de cor, por exemplo, 'red'. O argumento *fill* é opcional.

Linhas

O método line(*xy*, *fill*, *width*) desenha uma linha ou uma série de linhas. *xy* é uma lista de tuplas, como [(x, y), (x, y), ...], ou uma lista de inteiros, como [x1, y1, x2, y2, ...]. Cada ponto corresponde a um dos pontos que conectam as linhas que você está desenhando. O argumento *fill* opcional é a cor das linhas na forma de uma tupla RGBA ou o nome da cor. O argumento *width* opcional corresponde à largura das linhas, e o default será 1 se não for especificado.

Retângulos

O método rectangle(*xy*, *fill*, *outline*) desenha um retângulo. O argumento *xy* é uma tupla que representa uma caixa no formato (*left*, *top*, *right*, *bottom*). Os valores *left* e *top* especificam as coordenadas x e y do canto superior esquerdo do retângulo, enquanto *right* e *bottom* especificam o canto inferior direito. O argumento *fill* opcional corresponde à cor que preencherá a parte interna do retângulo. O argumento *outline* opcional é a cor do contorno do retângulo.

Elipses

O método ellipse(*xy*, *fill*, *outline*) desenha uma elipse. Se a largura e a altura da elipse forem idênticas, esse método desenhará um círculo. O argumento *xy* é uma tupla que representa uma caixa (*left*, *top*, *right*, *bottom*) e corresponde à caixa que contém exatamente a elipse. O argumento *fill* opcional é a cor da parte interna da elipse e o argumento *outline* opcional é a cor do contorno da elipse.

Polígonos

O método polygon(*xy*, *fill*, *outline*) desenha um polígono qualquer. O argumento *xy* é uma lista de tuplas, por exemplo, [(x, y), (x, y), ...], ou inteiros como [x1, y1, x2, y2, ...], que representa os pontos que conectam os lados do polígono. O último par de coordenadas será automaticamente conectado ao primeiro par. O argumento *fill* opcional é a cor da parte interna do polígono e o argumento *outline* opcional é a cor do contorno do polígono.

Exemplo de desenho

Digite o seguinte no shell interativo:

```
  >>> from PIL import Image, ImageDraw
  >>> im = Image.new('RGBA', (200, 200), 'white')
  >>> draw = ImageDraw.Draw(im)
u >>> draw.line([(0, 0), (199, 0), (199, 199), (0, 199), (0, 0)], fill='black')
v >>> draw.rectangle((20, 30, 60, 60), fill='blue')
w >>> draw.ellipse((120, 30, 160, 60), fill='red')
x >>> draw.polygon(((57, 87), (79, 62), (94, 85), (120, 90), (103, 113)), fill='brown')
y >>> for i in range(100, 200, 10):
          draw.line([(i, 0), (200, i - 100)], fill='green')

  >>> im.save('drawing.png')
```

Após criar um objeto Image com uma imagem branca de 200 × 200, passá-lo para ImageDraw.Draw() a fim de obter um objeto ImageDraw e armazenar esse objeto em draw, podemos chamar os métodos de desenho em draw. Nesse caso, criamos um contorno fino e preto nas bordas da imagem u, desenhamos um retângulo azul com seu canto superior esquerdo em (20, 30) e o canto inferior direito em (60, 60) v, uma elipse vermelha definida por uma caixa de (120, 30) a (160, 60) w, um polígono marrom com cinco pontos x e um padrão com linhas verdes desenhado com um loop for y. O arquivo *drawing.png* resultante terá a aparência mostrada na figura 17.14.

Há vários outros métodos para desenhar formas em objetos ImageDraw. A documentação completa está disponível em *http://pillow.readthedocs.org/en/latest/reference/ImageDraw.html*.

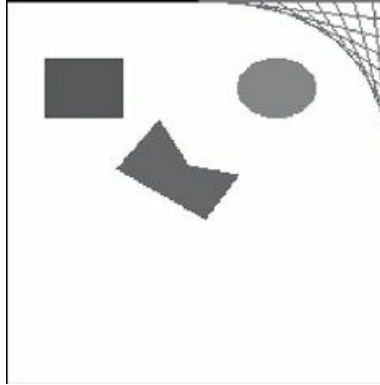



Figura 17.14 – A imagem drawing.png resultante.

Desenhando textos

O objeto ImageDraw também tem um método text() para desenhar um texto em uma imagem. O método text() aceita quatro argumentos: *xy*, *text*, *fill* e *font*.

- O argumento *xy* é uma tupla de dois inteiros que especifica o canto superior esquerdo da caixa de texto.
- O argumento *text* é a string com o texto que você quer escrever.
- O argumento *fill* opcional é a cor do texto.
- O argumento *font* opcional é um objeto ImageFont usado para definir o tipo de fonte e o tamanho do texto. Isso será descrito com mais detalhes a seguir.

Como geralmente é difícil saber com antecedência o tamanho de um bloco de texto para uma dada fonte, o módulo ImageDraw também disponibiliza um método textsize(). Seu primeiro argumento é a string de texto que você quer que seja dimensionado e o segundo argumento é um objeto ImageFont opcional. O método textsize() então retornará uma tupla de dois inteiros com a largura e a altura que esse texto terá na fonte especificada se for escrito na imagem. Você poderá usar essa largura e essa altura para ajudar a calcular exatamente em que ponto você quer colocar o texto em sua imagem.

Os primeiros três argumentos de text() são simples. Antes de usar text() para desenhar textos sobre uma imagem, vamos dar uma olhada no quarto argumento opcional, que é um objeto ImageFont.

Tanto text() quanto textsize() aceitam um objeto ImageFont opcional como último argumento. Para criar um desses objetos, inicialmente execute o

seguinte:

```
>>> from PIL import ImageFont
```

Agora que importamos o módulo ImageFont do Pillow, podemos chamar a função ImageFont.truetype(), que aceita dois argumentos. O primeiro argumento é uma string contendo o *arquivo TrueType* da fonte – é o arquivo da fonte que está em seu disco rígido. Um arquivo TrueType tem extensão *.ttf* e, normalmente, pode ser encontrado nas seguintes pastas:

- no Windows – *C:\Windows\Fonts*
- no OS X – */Library/Fonts* e */System/Library/Fonts*
- no Linux – */usr/share/fonts/truetype*

Não é preciso fornecer esses paths como parte da string com o nome do arquivo TrueType, pois o Python sabe que deve procurar automaticamente as fontes nesses diretórios. Porém o Python exibirá um erro se não puder encontrar a fonte que você especificar.

O segundo argumento de ImageFont.truetype() é um inteiro com o tamanho da fonte em *pontos* (e não, por exemplo, em pixels). Tenha em mente que o Pillow cria imagens PNG que têm 72 pixels por polegada (uma polegada = 2,54 cm), por padrão, e um ponto corresponde a 1/72 de uma polegada.

Digite o seguinte no shell interativo, substituindo FONT_FOLDER pelo nome da pasta utilizado pelo seu sistema operacional:

```
  >>> from PIL import Image, ImageDraw, ImageFont
  >>> import os
u >>> im = Image.new('RGBA', (200, 200), 'white')
v >>> draw = ImageDraw.Draw(im)
w >>> draw.text((20, 150), 'Hello', fill='purple')
  >>> fontsFolder = 'FONT_FOLDER' # por exemplo, '/Library/Fonts'
x >>> arialFont = ImageFont.truetype(os.path.join(fontsFolder, 'arial.ttf'), 32)
y >>> draw.text((100, 150), 'Howdy', fill='gray', font=arialFont)
  >>> im.save('text.png')
```

Após importar Image, ImageDraw, ImageFont e os, criamos um objeto Image para uma nova imagem branca de 200 × 200 u e criamos um objeto ImageDraw a partir do objeto Image v. Usamos text() para desenhar *Hello* em (20, 150) na cor roxa w. Não passamos o quarto argumento opcional nessa chamada a

text(), portanto o tipo de fonte e o tamanho desse texto não serão personalizados.

Para definir um tipo de fonte e o tamanho, inicialmente armazene o nome da pasta (por exemplo, *Library/Fonts*) em fontsFolder. Em seguida, chame ImageFont.truetype() passando-lhe o arquivo *.ttf* da fonte que você deseja usar, seguido de um inteiro para o tamanho da fonte x. Armazene o objeto Font obtido de ImageFont.truetype() em uma variável como arialFont e então passe essa variável para text() no último argumento nomeado. A chamada a text() em y desenha *Howdy* na posição (100, 150) em cinza, usando Arial com 32 pontos.

O arquivo *text.png* resultante terá a aparência mostrada na figura 17.15.

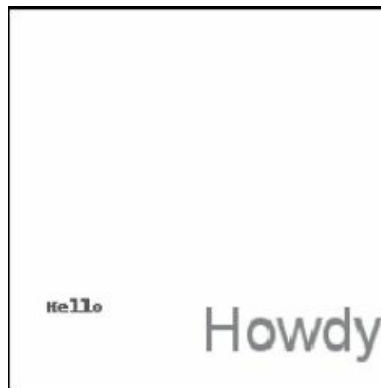



Figura 17.15 – A imagem text.png resultante.

Resumo

As imagens são constituídas de uma coleção de pixels; cada pixel tem um valor RGBA para sua cor e é acessível por meio de coordenadas x e y. Dois formatos comuns de imagens são o JPEG e o PNG. O módulo pillow é capaz de lidar com esses dois formatos de imagens, além de outros.

Quando uma imagem é carregada em um objeto Image, as dimensões correspondentes à sua largura e à sua altura são armazenadas como uma tupla de dois inteiros no atributo size. Os objetos do tipo de dado Image também têm métodos para manipulações comuns de imagem: crop(), copy(), paste(), resize(), rotate() e transpose(). Para salvar o objeto Image em um arquivo de imagem, chame o método save().

Se quiser que seu programa desenhe formas em uma imagem, utilize os

métodos de ImageDraw para desenhar pontos, linhas, retângulos, elipses e polígonos. O módulo também disponibiliza métodos para desenhar texto em um tipo de fonte, com um tamanho de sua escolha.

Embora aplicativos sofisticados (e caros) como o Photoshop ofereçam recursos de processamento em batch automáticos, você poderá usar scripts Python para realizar muitas das mesmas alterações gratuitamente. Nos capítulos anteriores, criamos programas Python que lidavam com arquivos em formato texto simples, planilhas, PDFs e outros formatos. Com o módulo pillow, ampliamos sua capacidade de programação para que você possa processar imagens também!

Exercícios práticos

1. O que é um valor RGBA?
2. Como podemos obter o valor RGBA de 'CornflowerBlue' com o módulo Pillow?
3. O que é uma tupla de caixa (box tuple)?
4. Qual função retorna um objeto Image, por exemplo, para um arquivo de imagem chamado *zophie.png*?
5. Como podemos descobrir a largura e a altura da imagem em um objeto Image?
6. Qual método deve ser chamado para que um objeto Image de uma imagem de 100 × 100 seja obtido, excluindo a quarta parte no canto inferior esquerdo?
7. Após fazer alterações em um objeto Image, como podemos salvá-lo em um arquivo de imagem?
8. Qual módulo contém o código para desenhar formas no Pillow?
9. Os objetos Image não têm métodos de desenho. Que tipo de objeto tem esses métodos? Como esse tipo de objeto pode ser obtido?

Projetos práticos

Para exercitar, escreva programas que façam as tarefas a seguir.

Estendendo e corrigindo os programas do projeto do capítulo

O programa *resizeAndAddLogo.py* deste capítulo trabalha com arquivos PNG e JPEG, porém o Pillow suporta muito mais formatos além desses dois. Estenda *resizeAndAddLogo.py* para que ele acesse imagens GIF e BMP também.

Outro pequeno problema está no fato de o programa modificar os arquivos PNG e JPEG somente se suas extensões de arquivo estiverem definidas com letras minúsculas. Por exemplo, o programa processará *zophie.png*, mas não *zophie.PNG*. Altere o código para que a verificação das extensões dos arquivos não leve em conta a diferença entre letras maiúsculas e minúsculas.

Por fim, o logo adicionado no canto inferior direito deve ser apenas uma pequena marca, porém, se a imagem tiver aproximadamente o mesmo tamanho do próprio logo, o resultado será semelhante ao apresentado na figura 17.16. Modifique *resizeAndAddLogo.py* para que a imagem tenha pelo menos duas vezes a largura e a altura da imagem do logo antes que o logo seja colado. Caso contrário, o logo não deverá ser adicionado.

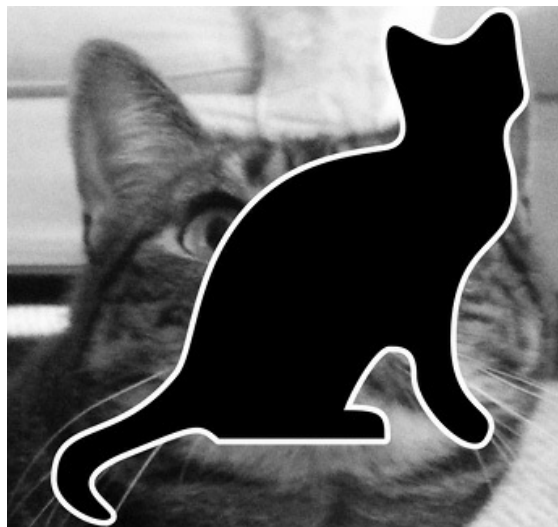



Figura 17.16 – Quando a imagem não for muito maior que o logo, o resultado não terá uma boa aparência.

Identificando pastas com fotos no disco rígido

Tenho o péssimo hábito de transferir arquivos de minha câmera digital para

pastas temporárias em algum lugar do disco rígido e depois me esqueço dessas pastas. Seria interessante criar um programa que pudesse varrer todo o disco rígido e encontrasse essas "pastas de fotos" deixadas no disco.

Crie um programa que acesse todas as pastas de seu disco rígido e encontre possíveis pastas de fotos. É claro que, inicialmente, você deverá definir o que será considerada uma "pasta de fotos"; vamos supor que seja qualquer pasta em que mais da metade dos arquivos sejam fotos. Como podemos definir quais arquivos correspondem a fotos? Em primeiro lugar, um arquivo de foto deve ter a extensão *.png* ou *.jpg*. Além disso, as fotos são imagens grandes; a largura e a altura de um arquivo de foto devem ter mais de 500 pixels. Isso serve como proteção, pois a maioria das fotos de câmeras digitais tem vários milhares de pixels de largura e de altura.

Como dica, apresentamos a seguir um esqueleto básico de como deverá ser a aparência desse programa:

```
#! python3
# Importe módulos e escreva comentários para descrever esse programa.

for foldername, subfolders, filenames in os.walk('C:\\'):
    numPhotoFiles = 0
    numNonPhotoFiles = 0
    for filename in filenames:
        # Verifica se a extensão do arquivo é diferente de .png e de .jpg.
        if TODO:
            numNonPhotoFiles += 1
            continue    # vai para o próximo nome de arquivo

        # Abre o arquivo de imagem usando o Pillow.

        # Verifica se a largura e a altura são maiores que 500.
        if TODO:
            # A imagem é grande o suficiente para ser considerada uma foto.
            numPhotoFiles += 1
        else:
            # A imagem é pequena demais para ser uma foto.
            numNonPhotoFiles += 1

    # Se mais da metade dos arquivos for composta de fotos,
```

```
# exibe o path absoluto da pasta.
if TODO:
    print(TODO)
```

Quando o programa for executado, ele deverá exibir o path absoluto de qualquer pasta de fotos na tela.

Cartões personalizados para indicar o assento

O capítulo 13 incluiu um projeto prático para criar convites personalizados a partir de uma lista de convidados em um arquivo em formato texto simples. Como projeto adicional, utilize o módulo pillow para criar imagens de cartões personalizados que indicarão onde seus convidados deverão se sentar. Para cada um dos convidados listados no arquivo *guests.txt* disponível nos recursos em *http://nostarch.com/automatestuff/*, gere um arquivo de imagem com o nome do convidado e algumas decorações floridas. Uma imagem de flores de domínio público está disponível nos recursos em *http://nostarch.com/automatestuff/*.

Para garantir que todos os cartões que indiquem o assento terão o mesmo tamanho, adicione um retângulo preto nas bordas da imagem para que, quando ela for impressa, haja uma linha indicando o local para recortar. Os arquivos PNG gerados pelo Pillow são definidos com 72 pixels por polegada (uma polegada = 2,54 cm), portanto um cartão de 4 × 5 polegadas exigirá uma imagem de 288 × 360 pixels.

CAPÍTULO 18

CONTROLANDO O TECLADO E O MOUSE COM AUTOMAÇÃO DE GUI

Conhecer vários módulos Python para editar planilhas, fazer download de arquivos e iniciar programas é útil, porém, às vezes, simplesmente não haverá módulo nenhum para as aplicações com as quais você deverá trabalhar. As ferramentas definitivas para automatizar tarefas em seu computador são os programas escritos por você que controlem diretamente o teclado

e o mouse. Esses programas podem controlar outras aplicações enviando-lhes pressionamentos de teclas e cliques de mouse virtuais, como se você estivesse diante de seu computador interagindo com essas aplicações. Essa técnica é conhecida como *automação de GUI* (Graphical User Interface, ou Interface gráfica de usuário). Com a automação de GUI, seus programas podem fazer tudo que um usuário humano diante do computador faz, exceto derramar café no teclado.

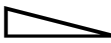
Pense na automação de GUI como na programação de um braço robótico. O braço robótico pode ser programado para digitar em seu teclado e mover o mouse para você. Essa técnica é particularmente útil para tarefas que envolvam muitos cliques automáticos ou o preenchimento de formulários.

O módulo pyautogui contém funções para simular movimentos do mouse, clicar em botões e girar a roda do mouse. Este capítulo discute somente um subconjunto dos recursos do PyAutoGUI; a documentação completa pode ser acessada em *http://pyautogui.readthedocs.org/*.

Instalando o módulo pyautogui

O módulo pyautogui pode enviar pressionamentos de tecla e cliques de mouse virtuais no Windows, no OS X e no Linux. Conforme o sistema operacional que você estiver usando, poderá ser necessário instalar outros módulos (chamados de *dependências*) antes de o PyAutoGUI ser instalado.

- No Windows, não há nenhum outro módulo para instalar.

- No OS X, execute **sudo pip3 install pyobjc-framework-Quartz**, **sudo pip3 install pyobjc-core** e, em seguida, **sudo pip3 install pyobjc**.
- No Linux, execute **sudo pip3 install python3-xlib**, **sudo apt-get install scrot**, **sudo apt-get install python3-tk**, e **sudo apt-get install python3-dev**. (O Scrot é um programa de captura de tela usado pelo PyAutoGUI.)

Após essas dependências terem sido instaladas, execute **pip install pyautogui** (ou **pip3** no OS X e no Linux) para instalar o PyAutoGUI.

O apêndice A tem informações completas sobre a instalação de módulos de terceiros. Para testar se o PyAutoGUI foi instalado corretamente, execute **import pyautogui** no shell interativo e verifique se houve alguma mensagem de erro.

Permanecendo no caminho certo

Antes de mergulhar de cabeça em uma automação de GUI, você deverá aprender a contornar os problemas que possam surgir. O Python pode mover o seu mouse e digitar teclas a uma velocidade incrível. Com efeito, ele poderá ser rápido demais para que os demais programas o acompanhem. Além disso, se algo der errado, mas seu programa continuar movendo o mouse, será difícil dizer o que o programa está fazendo exatamente ou recuperar-se do problema. Como as vassouras encantadas do filme *O aprendiz de feiticeiro* da Disney, em que o balde do Mickey continuava enchendo – e depois derramando –, seu programa poderá sair de controle, apesar de estar seguindo suas instruções com perfeição. Interromper o programa poderá ser difícil se o mouse estiver se movendo sozinho, impedindo que você clique na janela do IDLE para fechá-la. Felizmente, há diversas maneiras de evitar problemas de automação de GUI ou recuperar-se deles.

Encerrando tudo ao fazer logout

Talvez a maneira mais simples de interromper um programa com automação de GUI que esteja fora de controle seja fazer logout, o que encerrará todos os programas em execução. No Windows e no Linux, o atalho de teclado para fazer logout é CTRL-ALT-DEL. No OS X, é ⌘-SHIFT-OPTION-Q. Ao fazer logout, você perderá todo trabalho que não tenha sido salvado, porém, ao

menos, não precisará esperar uma reinicialização completa do computador.

Pausas e falhas com segurança

Você pode dizer ao seu script para esperar após cada chamada de função, o que proporcionará uma breve janela que possibilitará assumir o controle do mouse e do teclado caso algo dê errado. Para isso, defina a variável pyautogui.PAUSE com o número de segundos que você quer que dure a pausa. Por exemplo, após definir pyautogui.PAUSE = 1.5, toda chamada de função de PyAutoGUI fará uma pausa de um segundo e meio após realizar sua ação. As instruções que não pertencem ao PyAutoGUI não terão essa pausa.

O PyAutoGUI também tem um recurso de falha com segurança (fail safe). Ao mover o cursor do mouse para o canto superior esquerdo da tela, o PyAutoGUI lançará a exceção pyautogui.FailSafeException. Seu programa poderá tratar essa exceção com as instruções try e except ou poderá deixar que a exceção provoque uma falha em seu programa. De qualquer modo, o recurso de falha com segurança interromperá o programa se você mover rapidamente o mouse o máximo que conseguir para o canto superior esquerdo. Esse recurso pode ser desabilitado ao configurar pyautogui.FAILSAFE = False. Digite o seguinte no shell interativo:

```
>>> import pyautogui
>>> pyautogui.PAUSE = 1
>>> pyautogui.FAILSAFE = True
```

Nesse exemplo, importamos pyautogui e definimos pyautogui.PAUSE com 1 para que haja uma pausa de um segundo após cada chamada de função. Definimos pyautogui.FAILSAFE com True para habilitar o recurso de falha com segurança.

Controlando os movimentos do mouse

Nesta seção, aprenderemos a mover o mouse e a monitorar sua posição na tela usando o PyAutoGUI, porém, antes disso, será necessário entender de que modo o PyAutoGUI trabalha com as coordenadas.

As funções de mouse do PyAutoGUI utilizam as coordenadas x e y. A

figura 18.1 mostra o sistema de coordenadas da tela do computador; ele é semelhante ao sistema de coordenadas usado para imagens discutido no capítulo 17. A *origem*, em que tanto *x* quanto *y* são iguais a zero, está no canto superior esquerdo da tela. As coordenadas x aumentam para a direita e as coordenadas y, para baixo. Todas as coordenadas são números inteiros positivos; não há coordenadas negativas.

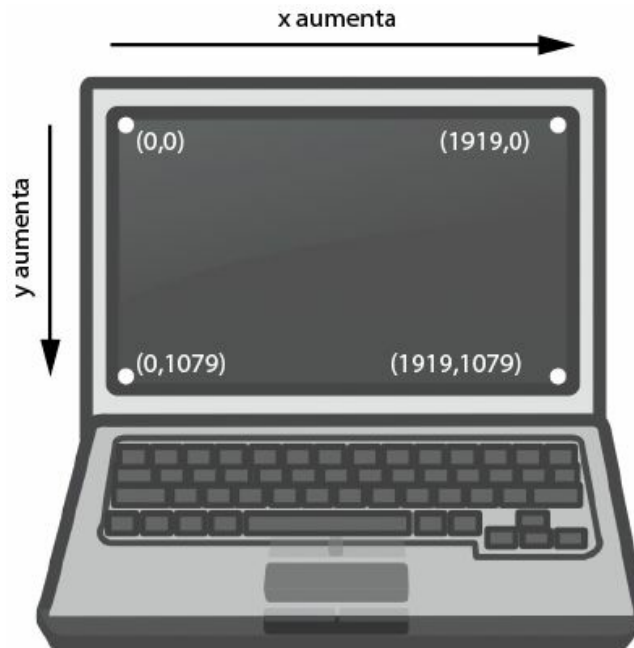



Figura 18.1 – As coordenadas de uma tela de computador com resolução de 1.920 × 1.080.

Sua *resolução* corresponde à quantidade de pixels da largura e da altura de sua tela. Se a resolução de sua tela estiver definida com 1.920 × 1.080, a coordenada do canto superior esquerdo será (0, 0) e a coordenada do canto inferior direito será (1919, 1079).

A função pyautogui.size() retorna uma tupla de dois inteiros que corresponde à largura e à altura da tela em pixels. Digite o seguinte no shell interativo:

```
>>> import pyautogui
>>> pyautogui.size()
(1920, 1080)
>>> width, height = pyautogui.size()
```

pyautogui.size() retorna (1920, 1080) em um computador com uma resolução de 1.920 x 1.080; conforme a resolução de sua tela, seu valor de retorno poderá

ser diferente. Podemos armazenar a largura e a altura de pyautogui.size() em variáveis como width e height para melhorar a legibilidade dos programas.

Movendo o mouse

Agora que você já sabe como funcionam as coordenadas da tela, vamos mover o mouse. A função pyautogui.moveTo() moverá instantaneamente o cursor do mouse para uma posição especificada na tela. Valores inteiros para as coordenadas x e y compõem o primeiro e o segundo argumento da função, respectivamente. Um argumento nomeado duration opcional, que pode ser um inteiro ou um número de ponto flutuante, especifica a quantidade de segundos que o mouse deverá demorar para deslocar-se até o destino. Se esse valor não for especificado, o default será 0 para que o movimento seja instantâneo. (Todos os argumentos nomeados duration nas funções do PyAutoGUI são opcionais.) Digite o seguinte no shell interativo:

```
>>> import pyautogui
>>> for i in range(10):
      pyautogui.moveTo(100, 100, duration=0.25)
      pyautogui.moveTo(200, 100, duration=0.25)
      pyautogui.moveTo(200, 200, duration=0.25)
      pyautogui.moveTo(100, 200, duration=0.25)
```

Nesse exemplo, o cursor do mouse se move em sentido horário em um padrão quadrangular entre as quatro coordenadas especificadas, em um total de dez vezes. Cada movimento dura um quarto de segundo, conforme especificado pelo argumento nomeado duration=0.25. Se você não tivesse passado um terceiro argumento em nenhuma das chamadas a pyautogui.moveTo(), o cursor do mouse teria sido "teletransportado" imediatamente de um ponto a outro.

A função pyautogui.moveRel() move o cursor do mouse *em relação à* sua posição atual. O exemplo a seguir move o mouse no mesmo padrão quadrangular, exceto pelo fato de iniciar o quadrado no ponto em que o mouse estiver na tela quando o código começar a executar:

```
>>> import pyautogui
>>> for i in range(10):
      pyautogui.moveRel(100, 0, duration=0.25)
```

```
pyautogui.moveRel(0, 100, duration=0.25)
pyautogui.moveRel(-100, 0, duration=0.25)
pyautogui.moveRel(0, -100, duration=0.25)
```

pyautogui.moveRel() também aceita três argumentos: a quantidade de pixels para mover-se horizontalmente à direita, a quantidade de pixels para mover-se verticalmente para baixo e (opcionalmente) o tempo que o movimento completo deverá demorar. Um inteiro negativo no primeiro e no segundo argumento fará o mouse mover-se para a esquerda ou para cima, respectivamente.

Obtendo a posição do mouse

Podemos determinar a posição atual do mouse ao chamar a função pyautogui.position(), que retorna uma tupla com as posições *x* e *y* do cursor do mouse no momento em que a função é chamada. Digite o seguinte no shell interativo, movendo o mouse após cada chamada:

```
>>> pyautogui.position()
(311, 622)
>>> pyautogui.position()
(377, 481)
>>> pyautogui.position()
(1536, 637)
```

É claro que seus valores de retorno variarão de acordo com a posição em que estiver o cursor de seu mouse.

Projeto: "Onde está o mouse neste momento?"

Ser capaz de determinar a posição do mouse é uma parte importante da criação de seus scripts com automação de GUI. Entretanto é quase impossível descobrir as coordenadas exatas de um pixel somente olhando para a tela. Será conveniente ter um programa que apresente constantemente as coordenadas x e y do cursor do mouse à medida que ele se mover.

De modo geral, eis o que o seu programa deverá fazer:

- Exibir as coordenadas x e y atuais do cursor do mouse.
- Atualizar essas coordenadas à medida que o mouse mover-se pela tela.

Isso significa que o seu código deverá fazer o seguinte:

- Chamar a função position() para acessar as coordenadas atuais.
- Apagar as coordenadas apresentadas anteriormente exibindo caracteres de backspace \b na tela.
- Tratar a exceção KeyboardInterrupt para que o usuário possa teclar CTRL-C para sair.

Abra uma nova janela no editor de arquivo e salve esse arquivo como *mouseNow.py*.

Passo 1: Importar o módulo

Inicie o seu programa com:

```
#! python3
# mouseNow.py – Exibe a posição atual do cursor do mouse.
import pyautogui
print('Press Ctrl-C to quit.')
#TODO: Obtém e exibe as coordenadas do mouse.
```

No início do programa, importamos o módulo pyautogui e exibimos um lembrete ao usuário informando que ele deve teclar CTRL-C para sair.

Passo 2: Criar o código para saída e o loop infinito

Um loop while infinito pode ser usado para exibir constantemente as coordenadas atuais do mouse obtidas de mouse.position(). Para o código que trata o encerramento do programa, será necessário capturar a exceção KeyboardInterrupt gerada sempre que o usuário pressionar CTRL-C. Se essa exceção não for tratada, um traceback pouco elegante e uma mensagem de erro serão exibidos ao usuário. Acrescente o seguinte em seu programa:

```
#! python3
# mouseNow.py – Exibe a posição atual do cursor do mouse.
import pyautogui
print('Press Ctrl-C to quit.')
try:
    while True:
        #TODO: Obtém e exibe as coordenadas do mouse.
u except KeyboardInterrupt:
```

```
v    print('\nDone.')
```

Para tratar a exceção, insira o loop while infinito em uma instrução try. Quando o usuário pressionar CTRL-C, a execução do programa será desviada para a cláusula except u, e Done. será exibido em uma nova linha v.

Passo 3: Obter e exibir as coordenadas do mouse

O código no loop while deve obter as coordenadas atuais do mouse, formatá-las para que tenham uma aparência elegante e exibi-las. Adicione o código a seguir no loop while:

```
#! python3
# mouseNow.py – Exibe a posição atual do cursor do mouse.
import pyautogui
print('Press Ctrl-C to quit.')
--trecho removido--
        # Obtém e exibe as coordenadas do mouse.
        x, y = pyautogui.position()
        positionStr = 'X: ' + str(x).rjust(4) + ' Y: ' + str(y).rjust(4)
--trecho removido--
```

Ao usar o truque da atribuição múltipla, as variáveis x e y recebem os valores dos dois inteiros da tupla retornada por pyautogui.position(). Ao passar x e y à função str(), podemos obter as formas em string das coordenadas cujos valores são inteiros. O método rjust() justificará as strings à direita para que elas ocupem o mesmo espaço, independentemente de a coordenada ter um, dois, três ou quatro dígitos. Concatenar as strings das coordenadas justificadas à direita com os rótulos 'X:' e ' Y: ' resulta em uma string elegantemente formatada que será armazenada em positionStr.

No final de seu programa, acrescente o código a seguir:

```
#! python3
# mouseNow.py – Exibe a posição atual do cursor do mouse.
--trecho removido--
        print(positionStr, end='')
u       print('\b' * len(positionStr), end='', flush=True)
```

Essas instruções exibem positionStr na tela. O argumento nomeado end='' de print() evita que o caractere default de quebra de linha seja adicionado no final

da linha exibida. É possível apagar o texto já apresentado na tela – mas somente para a linha de texto mais recente. Após ter exibido um caractere de quebra de linha, não será mais possível apagar nenhum dado exibido antes dele.

Para apagar um texto, exiba o caractere de escape \b referente a um backspace. Esse caractere especial apaga um caractere no final da linha atual da tela. A linha em u utiliza repetição de string para gerar uma string com a quantidade de caracteres \b igual ao tamanho da string armazenada em positionStr, o que terá o efeito de apagar a string positionStr exibida previamente.

Por motivos técnicos que estão além do escopo deste livro, sempre passe flush=True às chamadas a print() que exibam caracteres \b de backspace. Caso contrário, o texto na tela poderá não ser atualizado conforme desejado.

Como o loop while é repetido rapidamente, o usuário não perceberá que você está apagando e exibindo novamente o número completo na tela. Por exemplo, se a coordenada x for 563 e o mouse mover-se um pixel para a direita, parecerá que somente o 3 em 563 foi alterado para 4.

Ao executar o programa, somente duas linhas serão exibidas. Elas terão o seguinte aspecto:

```
Press Ctrl-C to quit.
X:  290 Y:  424
```

A primeira linha exibe a instrução para pressionar CTRL-C para sair do programa. A segunda linha com as coordenadas do mouse mudará à medida que você mover o mouse pela tela. Ao usar esse programa, você poderá descobrir as coordenadas do mouse em seus scripts com automação de GUI.

Controlando a interação com o mouse

Agora que você já sabe mover o mouse e descobrir em que ponto ele está na tela, você está pronto para começar a clicar, arrastar e fazer rolagens.

Clicando o mouse

Para enviar um clique de mouse virtual ao seu computador, chame o método

pyautogui.click(). Por padrão, esse clique utiliza o botão esquerdo do mouse e ocorre sempre no local em que o cursor do mouse estiver posicionado no momento. Podemos passar as coordenadas x e y do clique como o primeiro e o segundo parâmetros opcionais se você quiser que o clique ocorra em outro local que não seja a posição atual do mouse.

Se quiser especificar o botão do mouse a ser usado, inclua o argumento nomeado button com um valor igual a 'left', 'middle' ou 'right'. Por exemplo, pyautogui.click(100, 150, button='left') clicará o botão esquerdo do mouse nas coordenadas (100, 150), enquanto pyautogui.click(200, 250, button='right') fará um clique com o botão direito do mouse em (200, 250).

Digite o seguinte no shell interativo:

```
>>> import pyautogui
>>> pyautogui.click(10, 5)
```

Você deverá ver o ponteiro do mouse mover-se para próximo do canto superior esquerdo de sua tela e clicar uma vez. Um "clique" completo é definido como o pressionamento do botão do mouse e a sua liberação sem mover o cursor. Um clique também pode ser feito por meio de uma chamada a pyautogui.mouseDown(), que somente pressionará o botão do mouse, e a pyautogui.mouseUp(), que somente soltará o botão. Essas funções têm os mesmos argumentos que click() e, na verdade, a função click() é somente um wrapper conveniente em torno dessas duas chamadas de função.

Como uma conveniência adicional, a função pyautogui.doubleClick() executará dois cliques com o botão esquerdo do mouse, enquanto as funções pyautogui.rightClick() e pyautogui.middleClick() executarão um clique com os botões direito e central do mouse, respectivamente.

Arrastando o mouse

Arrastar quer dizer mover o mouse enquanto um de seus botões estiver pressionado. Por exemplo, podemos mover arquivos entre pastas ao arrastar os ícones das pastas, ou podemos mover compromissos em um aplicativo de calendário.

O PyAutoGUI disponibiliza as funções pyautogui.dragTo() e pyautogui.dragRel() para arrastar o cursor do mouse para uma nova posição ou para uma posição

relativa à sua localização atual. Os argumentos de dragTo() e de dragRel() são os mesmos de moveTo() e de moveRel(): a coordenada x/movimento horizontal, a coordenada y/movimento vertical e um tempo de duração opcional. (O OS X não arrasta o mouse de forma correta quando ele é movido muito rapidamente, portanto passar um argumento nomeado duration é recomendável.)

Para testar essas funções, abra um aplicativo gráfico de desenhos como o Paint no Windows, o Paintbrush no OS X ou o GNU Paint no Linux. (Se você não tiver um aplicativo de desenho, utilize um online, disponível em *http://sumopaint.com/*.) Utilizarei o PyAutoGUI para desenhar nesses aplicativos.

Com o cursor do mouse sobre a tela do aplicativo de desenho e a ferramenta Pencil (Lápis) ou Brush (Pincel) selecionada, digite o código a seguir em uma nova janela do editor de arquivo e salve o programa como *spiralDraw.py*:

```
  import pyautogui, time
u time.sleep(5)
v pyautogui.click()    # clica para deixar o foco no programa de desenho
  distance = 200
  while distance > 0:
w     pyautogui.dragRel(distance, 0, duration=0.2)   # move para a direita
x     distance = distance - 5
y     pyautogui.dragRel(0, distance, duration=0.2)   # move para baixo
z     pyautogui.dragRel(-distance, 0, duration=0.2)  # move para a esquerda
      distance = distance - 5
      pyautogui.dragRel(0, -distance, duration=0.2)  # move para cima
```

Ao executar esse programa, haverá um atraso de cinco segundos u para você mover o cursor do mouse para a janela do programa de desenho com a ferramenta Pencil ou Brush selecionada. Em seguida, *spiralDraw.py* assumirá o controle do mouse e clicará para que o programa de desenho tenha o foco v. Uma janela estará em *foco* quando tiver um cursor ativo piscante, e as ações que você executar – por exemplo, digitar ou, nesse caso, arrastar o mouse – afetarão essa janela. Depois que o programa de desenho tiver o foco, *spiralDraw.py* desenhará um padrão de espiral quadrangular conforme mostrado na figura 18.2.

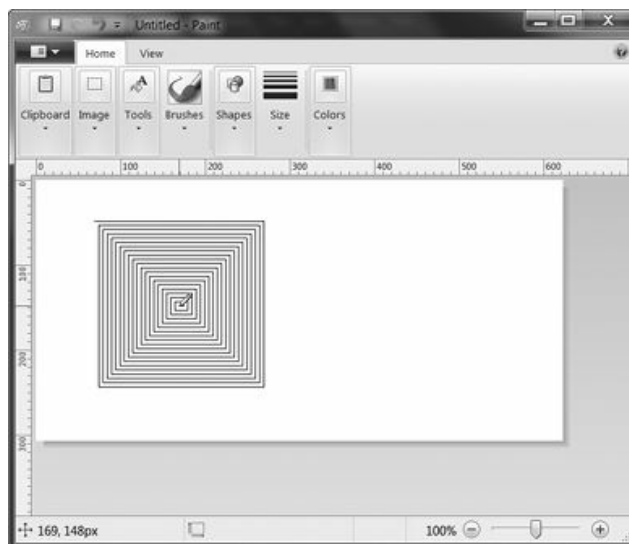



Figura 18.2 – O resultado do exemplo com pyautogui.dragRel().

A variável distance começa em 200, portanto, na primeira iteração do loop while, a chamada inicial a dragRel() arrasta o cursor 200 pixels para a direita e demora 0,2 segundo w. distance é decrementada para 195 x e a segunda chamada a dragRel() arrasta o cursor 195 pixels para baixo y. A terceira chamada a dragRel() arrasta o cursor -195 horizontalmente (195 para a esquerda) z, distance é decrementada para 190 e a última chamada a dragRel() arrasta o cursor 190 pixels para cima. A cada iteração, o mouse é arrastado para a direita, para baixo, para a esquerda e para cima, e distance será um pouco menor do que era na iteração anterior. Ao executar esse código em um loop, podemos mover o cursor do mouse para desenhar uma espiral quadrangular.

Você poderá desenhar essa espiral manualmente (ou com o mouse), porém deverá trabalhar lentamente para ter essa precisão. O PyAutoGUI pode fazer isso em alguns segundos!

NOTA Você poderia fazer o seu código desenhar a imagem usando as funções de desenho do módulo pillow – veja o capítulo 17 para obter mais informações. Porém usar a automação de GUI permite que você utilize as ferramentas sofisticadas de desenho disponibilizadas pelos programas gráficos, por exemplo, gradientes, pincéis diferentes ou preenchimento de cor.

Fazendo rolagens com o mouse

A última função de mouse de PyAutoGUI é scroll(), que recebe um argumento

inteiro para a quantidade de unidades segundo a qual o mouse fará rolagens para cima ou para baixo. O tamanho de uma unidade varia de acordo com cada sistema operacional e cada aplicação, portanto será necessário fazer experimentos para ver exatamente qual será a distância da rolagem em sua situação em particular. A rolagem ocorre na posição atual do cursor do mouse. Passar um inteiro positivo fará uma rolagem para cima, enquanto passar um inteiro negativo fará uma rolagem para baixo. Execute o código a seguir no shell interativo enquanto o cursor do mouse estiver na janela do IDLE:

```
>>> pyautogui.scroll(200)
```

Você verá o IDLE fazer uma breve rolagem para cima – e depois retornar. A rolagem para baixo ocorre porque o IDLE faz uma rolagem até o final automaticamente após executar uma instrução. Digite o código a seguir no lugar do código anterior:

```
>>> import pyperclip
>>> numbers = ''
>>> for i in range(200):
      numbers = numbers + str(i) + '\n'

>>> pyperclip.copy(numbers)
```

Esse código importa pyperclip e define uma string numbers vazia. O código então percorre 200 números em um loop e adiciona cada número a numbers, juntamente com um caractere de quebra de linha. Após pyperclip.copy(numbers) executar, o clipboard estará carregado com 200 linhas de números. Abra uma nova janela no editor de arquivo e cole o texto nessa janela. Isso dará a você uma janela de texto grande em que será possível fazer rolagens. Digite o código a seguir no shell interativo:

```
>>> import time, pyautogui
>>> time.sleep(5); pyautogui.scroll(100)
```

Na segunda linha, forneça dois comandos separados por ponto-e-vírgula, que dirá ao Python para executar os comandos como se eles estivessem em linhas separadas. A única diferença é que o shell interativo não pedirá uma entrada entre as duas instruções. Isso é importante nesse exemplo, pois

queremos que a chamada a pyautogui.scroll() ocorra automaticamente após a espera. (Observe que, embora inserir dois comandos em uma linha possa ser conveniente no shell interativo, em seus programas, você deve continuar inserindo cada instrução em uma linha separada.)

Após teclar ENTER para executar o código, você terá cinco segundos para clicar na janela do editor de arquivo e colocá-la em foco. Depois que a pausa se esgotar, a chamada a pyautogui.scroll() fará uma rolagem para cima na janela do editor de arquivo após o intervalo de cinco segundos.

Trabalhando com a tela

Seus programas com automação de GUI não precisam clicar nem digitar cegamente. O PyAutoGUI tem recursos de captura de tela para criar um arquivo de imagem com base no conteúdo atual da tela. Essas funções também podem retornar um objeto Image do Pillow com a aparência da tela atual. Se você não estiver seguindo este livro sequencialmente, leia o capítulo 17 e instale o módulo pillow antes de prosseguir com esta seção.

Em computadores Linux, o programa scrot deve ser instalado para que as funções de captura de tela do PyAutoGUI possam ser usadas. Em uma janela do Terminal, execute **sudo apt-get install scrot** para instalar esse programa. Se você estiver no Windows ou no OS X, pule esse passo e continue nesta seção.

Obtendo uma captura de tela

Para obter capturas de tela em Python, chame a função pyautogui.screenshot(). Digite o seguinte no shell interativo:

```
>>> import pyautogui
>>> im = pyautogui.screenshot()
```

A variável im conterá o objeto Image da captura de tela. Podemos agora chamar os métodos do objeto Image na variável im, como faríamos com qualquer objeto Image. Digite o seguinte no shell interativo:

```
>>> im.getpixel((0, 0))
(176, 176, 175)
>>> im.getpixel((50, 200))
```

```
(130, 135, 144)
```

Passe uma tupla de coordenadas, por exemplo, (0, 0) ou (50, 200), a getpixel() e essa função informará a cor do pixel nessas coordenadas de sua imagem. O valor de retorno de getpixel() é uma tupla RGB com três inteiros correspondentes à quantidade de vermelho, de verde e de azul no pixel. (Não há nenhum quarto valor para alpha, pois as imagens das capturas de tela são totalmente opacas.) É assim que seus programas poderão "ver" o que está na tela no momento.

Analisando a tela capturada

Suponha que um dos passos em seu programa com automação de GUI seja clicar em um botão cinza. Antes de chamar o método click(), você poderá obter uma captura de tela e observar o pixel em que o script está prestes a clicar. Se ele não tiver o mesmo cinza do botão cinza, seu programa saberá que há algo errado. Talvez a janela tenha sido movida inesperadamente ou, quem sabe, um diálogo pop-up tenha bloqueado o botão. A essa altura, em vez de continuar – e, possivelmente, causar uma enorme confusão ao clicar em algo errado –, seu programa "verá" que não está clicando no item correto e poderá interromper a si mesmo.

A função pixelMatchesColor() do PyAutoGUI retornará True se o pixel nas coordenadas x e y especificadas na tela corresponderem à cor fornecida. O primeiro e o segundo argumentos são inteiros para as coordenadas x e y, e o terceiro argumento é uma tupla com três inteiros para a cor RGB à qual o pixel da tela deve corresponder. Digite o seguinte no shell interativo:

```
  >>> import pyautogui
  >>> im = pyautogui.screenshot()
u >>> im.getpixel((50, 200))
  (130, 135, 144)
v >>> pyautogui.pixelMatchesColor(50, 200, (130, 135, 144))
  True
w >>> pyautogui.pixelMatchesColor(50, 200, (255, 135, 144))
  False
```

Após fazer uma captura de tela e usar getpixel() para obter uma tupla RGB com a cor de um pixel em coordenadas específicas u, passe as mesmas

coordenadas e a tupla RGB para pixelMatchesColor() v, que deverá retornar True. Em seguida, altere o valor da tupla RGB e chame pixelMatchesColor() novamente para as mesmas coordenadas w. Essa chamada deverá retornar False. Chamar esse método poderá ser útil sempre que seus programas com automação de GUI estiverem prestes a chamar click(). Observe que as cores nas coordenadas especificadas devem ser *exatamente* iguais. Se elas forem diferentes, mesmo que seja somente um pouco – por exemplo, (255, 255, 254) no lugar de (255, 255, 255) –, pixelMatchesColor() retornará False.

Projeto: Estendendo o programa mouseNow

Você pode estender o projeto anterior *mouseNow.py* deste capítulo para que ele forneça não só as coordenadas x e y da posição atual do cursor do mouse, mas também a cor RGB do pixel em que está o cursor. Modifique o código no loop while de *mouseNow.py* para que tenha o seguinte aspecto:

```
#! python3
# mouseNow.py – Exibe a posição atual do cursor do mouse.
--trecho removido--
        positionStr = 'X: ' + str(x).rjust(4) + ' Y: ' + str(y).rjust(4)
        pixelColor = pyautogui.screenshot().getpixel((x, y))
        positionStr += ' RGB: (' + str(pixelColor[0]).rjust(3)
        positionStr += ', ' + str(pixelColor[1]).rjust(3)
        positionStr += ', ' + str(pixelColor[2]).rjust(3) + ')'
        print(positionStr, end='')
--trecho removido--
```

Agora, quando *mouseNow.py* for executado, a saída incluirá o valor da cor RGB do pixel em que estiver o cursor do mouse.

```
Press Ctrl-C to quit.
X:  406 Y:   17 RGB: (161,  50,  50)
```

Essas informações, juntamente com a função pixelMatchesColor(), facilitarão adicionar verificações de cor de pixels em seus scripts com automação de GUI.

Reconhecimento de imagens

O que ocorreria, porém, se você não soubesse de antemão em que ponto o PyAutoGUI deve clicar? Você poderia usar o recurso de reconhecimento de imagens. Forneça uma imagem do que você quer clicar ao PyAutoGUI e deixe que ele descubra as coordenadas.

Por exemplo, se você capturou anteriormente uma tela com a imagem de um botão Submit (Submeter) em *submit.png*, a função locateOnScreen() retornará as coordenadas em que essa imagem se encontra. Para ver como locateOnScreen() funciona, experimente fazer a captura de uma pequena área de sua tela; em seguida, salve a imagem e digite o código a seguir no shell interativo, substituindo 'submit.png' pelo nome do arquivo que contém a sua captura de tela:

```
>>> import pyautogui
>>> pyautogui.locateOnScreen('submit.png')
(643, 745, 70, 29)
```

A tupla de quatro inteiros retornada por locateOnScreen() contém a coordenada x da borda esquerda, a coordenada y da borda superior, a largura e a altura do primeiro local na tela em que a imagem foi encontrada. Se você estiver testando isso em seu computador com a sua própria captura de tela, seu valor de retorno será diferente dos dados apresentados aqui.

Se a imagem não puder ser encontrada na tela, locateOnScreen() retornará None. Observe que a imagem na tela deve corresponder perfeitamente à imagem fornecida para que seja reconhecida. Caso a imagem tenha apenas um pixel de diferença, isso será suficiente para locateOnScreen() retornar None.

Se a imagem puder ser encontrada em diversos pontos da tela, locateAllOnScreen() retornará um objeto Generator que poderá ser passado a list() de modo que uma lista de tuplas de quatro inteiros possa ser retornada. Haverá uma tupla de quatro inteiros para cada posição em que a imagem for encontrada na tela. Prossiga com o exemplo no shell interativo digitando o seguinte (substituindo 'submit.png' pelo seu próprio arquivo de imagem):

```
>>> list(pyautogui.locateAllOnScreen('submit.png'))
[(643, 745, 70, 29), (1007, 801, 70, 29)]
```

Cada uma das tuplas de quatro inteiros representa uma área da tela. Se sua imagem for encontrada somente em uma área, o uso de list() e de

locateAllOnScreen() resultará em uma lista contendo apenas uma tupla.

De posse da tupla de quatro inteiros referente à área da tela em que sua imagem foi encontrada, você poderá clicar no centro dessa área ao passar a tupla para a função center(); essa função retorna as coordenadas x e y do centro dessa área. Digite o seguinte no shell interativo, substituindo os argumentos pelo seu próprio nome de arquivo, pela sua tupla de quatro inteiros e pelo seu par de coordenadas.

```
>>> pyautogui.locateOnScreen('submit.png')
(643, 745, 70, 29)
>>> pyautogui.center((643, 745, 70, 29))
(678, 759)
>>> pyautogui.click((678, 759))
```

Depois que tiver as coordenadas do centro retornadas por center(), passar essas coordenadas a click() fará um clique ser executado no centro da área da tela que corresponde à imagem passada para locateOnScreen().

Controlando o teclado

O PyAutoGUI também tem funções para enviar pressionamentos de teclas virtuais ao seu computador, o que possibilita preencher formulários ou inserir texto em aplicações.

Enviando uma string a partir do teclado

A função pyautogui.typewrite() envia pressionamentos de teclas virtuais ao computador. O que esses pressionamentos de tecla fazem depende da janela e do campo de texto que estiverem com o foco. Você poderá inicialmente enviar um clique de mouse para o campo de texto desejado a fim de garantir que ele tenha o foco.

Como um exemplo simples, vamos usar o Python para digitar automaticamente as palavras *Hello world!* em uma janela do editor de arquivo. Inicialmente, abra uma nova janela no editor de arquivo e posicione-a no canto superior esquerdo de sua tela para que o PyAutoGUI clique no lugar certo e dê o foco a ela. Em seguida, digite o seguinte no shell interativo.

```
>>> pyautogui.click(100, 100); pyautogui.typewrite('Hello world!')
```

Observe que, ao colocar dois comandos na mesma linha separados por ponto-e-vírgula, impedimos que o shell interativo solicite uma entrada entre a execução das duas instruções. Isso evita que você acidentalmente dê o foco a uma nova janela entre as chamadas a click() e a typewrite(), o que causaria confusão no exemplo.

O Python inicialmente enviará um clique de mouse virtual às coordenadas (100, 100), o que deverá enviar um clique à janela do editor de arquivo e a deixará em foco. A chamada a typewrite() enviará o texto *Hello world!* à janela, deixando-a conforme mostrada na figura 18.3. Agora você tem um código que digita por você!

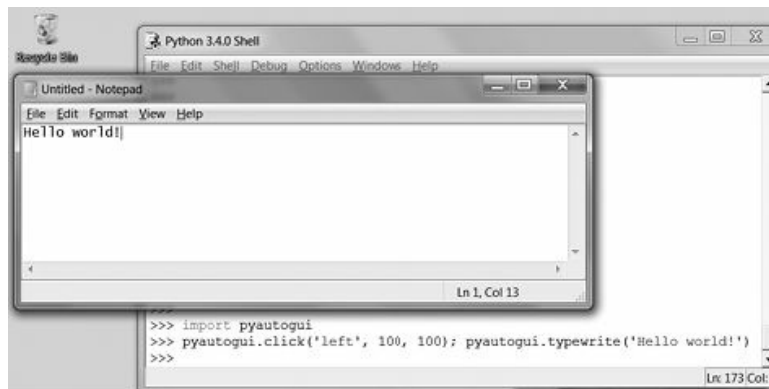



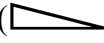
Figura 18.3 – Usando o PyAutogGUI para clicar na janela do editor de arquivo e digitar Hello world! *nessa janela.*

Por padrão, a função typewrite() digitará a string completa instantaneamente. No entanto podemos passar um segundo argumento opcional para acrescentar uma breve pausa entre cada caractere. Esse segundo argumento é um inteiro ou um número de ponto flutuante com a quantidade de segundos para a pausa. Por exemplo, pyautogui.typewrite('Hello world!', 0.25) provocará uma espera de um quarto de segundo após *H* ter sido digitado, outro quarto de segundo após *e* e assim sucessivamente. Esse efeito de digitação gradual poderá ser conveniente em aplicações mais lentas, que não sejam capazes de processar as teclas de forma rápida o suficiente para acompanhar o PyAutoGUI.

Para caracteres como *A* ou *!*, o PyAutoGUI também simulará automaticamente a tecla SHIFT sendo pressionada.

Nomes das teclas

Nem todas as teclas são facilmente representadas com um único caractere de texto. Por exemplo, como podemos representar a tecla SHIFT ou a seta para a esquerda como um único caractere? No PyAutoGUI, essas teclas são representadas por strings curtas: 'esc' para a tecla ESC ou 'enter' para a tecla ENTER.

Em vez de um único argumento de string, uma lista dessas strings de teclas poderá ser passada para typewrite(). Por exemplo, a chamada a seguir faz a tecla A ser pressionada, em seguida a tecla B, a tecla de seta para a esquerda duas vezes e, por fim, as teclas X e Y:

```
>>> pyautogui.typewrite(['a', 'b', 'left', 'left', 'X', 'Y'])
```

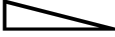
Como o pressionamento da tecla de seta para a esquerda faz o cursor do teclado mover-se, a saída dessa instrução será igual a *XYab*. A tabela 18.1 lista as strings de teclas do PyAutoGUI que podem ser passadas a typewrite() para simular o pressionamento de qualquer combinação de teclas.

Você também pode analisar a lista pyautogui.KEYBOARD_KEYS para conhecer todas as strings de teclas possíveis que o PyAutoGUI aceita. A string 'shift' refere-se à tecla SHIFT esquerda e é equivalente a 'shiftleft'. O mesmo se aplica às strings 'ctrl', 'alt' e 'win'; todas elas se referem às teclas do lado esquerdo.

Tabela 18.1 – Atributos de PyKeyboard

String da tecla	Significado
'a', 'b', 'c', 'A', 'B', 'C', '1', '2', '3', '!', '@', '#' e assim por diante	As teclas para os caracteres únicos
'enter' (ou 'return' ou '\n')	A tecla ENTER
'esc'	A tecla ESC
'shiftleft', 'shiftright'	As teclas SHIFT da esquerda e da direita
'altleft', 'altright'	As teclas ALT da esquerda e da direita
'ctrlleft', 'ctrlright'	As teclas CTRL da esquerda e da direita
'tab' (ou '\t')	A tecla TAB

'backspace', 'delete'	As teclas BACKSPACE e DELETE
'pageup', 'pagedown'	As teclas PAGE UP e PAGE DOWN
'home', 'end'	As teclas HOME e END
'up', 'down', 'left', 'right'	As teclas de direção para cima, para baixo, para a esquerda e para a direita
'f1', 'f2', 'f3' e assim por diante	As teclas de F1 a F12
'volumemute', 'volumedown', 'volumeup'	As teclas mute, de aumento de volume e de diminuição (alguns teclados não têm essas teclas, mas seu sistema operacional será capaz de entender esses pressionamentos de tecla simulados)
'pause'	A tecla PAUSE
'capslock', 'numlock', 'scrolllock'	As teclas CAPS LOCK, NUM LOCK e SCROLL LOCK
'insert'	A tecla INS ou INSERT
'printscreen'	A tecla PRTSC ou PRINT SCREEN
'winleft', 'winright'	As teclas WIN da esquerda e da direita (no Windows)
'command'	A tecla Command ( ) (no OS X)
'option'	A tecla OPTION (no OS X)

Pressionando e soltando as teclas

De modo muito semelhante às funções mouseDown() e mouseUp(), pyautogui.keyDown() e pyautogui.keyUp() enviarão pressionamentos e liberações de teclas virtuais ao computador. Essas funções recebem uma string de tecla (veja a tabela 18.1) como argumento. Por questões de conveniência, o PyAutoGUI disponibiliza a função pyautogui.press(), que chama ambas as funções para simular um pressionamento de tecla completo.

Execute o código a seguir, que digitará um caractere de cifrão (obtido ao segurar a tecla SHIFT e pressionar 4):

```
>>> pyautogui.keyDown('shift'); pyautogui.press('4'); pyautogui.keyUp('shift')
```

Essa linha pressiona SHIFT, pressiona (e solta) a tecla 4 e solta a tecla SHIFT. Se houver necessidade de digitar uma string em um campo de texto, a função typewrite() será mais adequada. Entretanto, em aplicações que aceitem comandos de uma única tecla, a função press() será a abordagem mais simples.

Combinações para atalhos de teclado

Uma *tecla de atalho* ou *atalho de teclado* (hotkey ou shortcut) é uma combinação de teclas para acionar uma função em um aplicativo. A tecla de atalho comum para copiar uma seleção é CTRL-C (no Windows e no Linux) ou -C (no OS X). O usuário deve pressionar e segurar a tecla CTRL, pressionar a tecla C e então soltar as teclas C e CTRL. Para fazer isso com as funções keyDown() e keyUp() do PyAutoGUI, digite o seguinte:

```
pyautogui.keyDown('ctrl')
pyautogui.keyDown('c')
pyautogui.keyUp('c')
pyautogui.keyUp('ctrl')
```

Isso é bem complicado. Em seu lugar, utilize a função pyautogui.hotkey(), que aceita vários argumentos de strings de teclas, pressiona essas teclas na sequência e as solta na ordem inversa. Para o exemplo com CTRL-C, o código será simplesmente:

```
pyautogui.hotkey('ctrl', 'c')
```

Essa função é especialmente útil para combinações maiores de teclas de atalho. No Word, a combinação de teclas CTRL-ALT-SHIFT-S exibe o painel Style (Estilos). Em vez de fazer oito diferentes chamadas de função (quatro chamadas para keyDown() e quatro chamadas para keyUp()), podemos simplesmente chamar hotkey('ctrl', 'alt', 'shift', 's').

Com uma nova janela do editor de arquivo do IDLE no canto superior esquerdo de sua tela, digite o seguinte no shell interativo (no OS X, substitua 'alt' por 'ctrl'):

```
>>> import pyautogui, time
>>> def commentAfterDelay():
u       pyautogui.click(100, 100)
v       pyautogui.typewrite('In IDLE, Alt-3 comments out a line.')
        time.sleep(2)
w       pyautogui.hotkey('alt', '3')

>>> commentAfterDelay()
```

Esse código define uma função commentAfterDelay(), que, quando chamada,

clicará na janela do editor de arquivo para que ela tenha o foco u, digitará *In IDLE, Atl-3 comments out a line* v, fará uma pausa de dois segundos e então simulará o pressionamento da tecla de atalho ALT-3 (ou CTRL-3 no OS X) w. Esse atalho de teclado adiciona dois caracteres # à linha atual, deixando-a comentada. (É interessante conhecer esse truque quando você estiver escrevendo seu próprio código no IDLE.)

Revisão das funções de PyAutoGUI

Como este capítulo abordou diversas funções diferentes, eis um breve resumo para servir de referência:

- **moveTo(*x*, *y*)** Move o cursor do mouse para as coordenadas *x* e *y* especificadas.
- **moveRel(*xOffset*, *yOffset*)** Move o cursor do mouse em relação à sua posição atual.
- **dragTo(*x*, *y*)** Move o cursor do mouse enquanto o botão esquerdo está pressionado.
- **dragRel(*xOffset*, *yOffset*)** Move o cursor do mouse em relação à sua posição atual enquanto o botão esquerdo está pressionado.
- **click(*x*, *y*, *button*)** Simula um clique (do botão esquerdo, por padrão).
- **rightClick()** Simula um clique do botão direito.
- **middleClick()** Simula um clique do botão do meio.
- **doubleClick()** Simula um clique duplo do botão esquerdo.
- **mouseDown(*x*, *y*, *button*)** Simula o pressionamento do botão especificado na posição *x*, *y*.
- **mouseUp(*x*, *y*, *button*)** Simula a liberação do botão especificado na posição *x*, *y*.
- **scroll(*units*)** Simula a roda do mouse. Um argumento positivo faz uma rolagem para cima; um argumento negativo faz uma rolagem para baixo.
- **typewrite(*message*)** Digita os caracteres da string com a mensagem especificada.
- **typewrite([*key1*, *key2*, *key3*])** Digita as teclas referentes às strings de tecla especificadas.

- **press(*key*)** Pressiona a tecla referente à string de tecla especificada.
- **keyDown(*key*)** Simula o pressionamento da tecla especificada.
- **keyUp(*key*)** Simula a liberação da tecla especificada.
- **hotkey([*key1*, *key2*, *key3*])** Simula o pressionamento das strings de tecla especificadas em sequência e, em seguida, a liberação dessas teclas na ordem inversa.
- **screenshot()** Retorna uma captura de tela na forma de um objeto Image. (Veja o capítulo 17 para obter informações sobre os objetos Image.)

Projeto: Preenchimento automático de formulários

De todas as tarefas maçantes, preencher formulários está entre as mais desagradáveis. Não é por acaso que agora, no último projeto de capítulo, você vá se livrar dela. Suponha que você tenha uma quantidade enorme de dados em uma planilha e precise digitar esses dados novamente na interface de formulário de outra aplicação – sem ter um estagiário que faça isso para você. Embora algumas aplicações tenham um recurso Import (Importar) que permita fazer o upload de uma planilha com as informações, às vezes, pode parecer que não há nenhuma outra maneira que não seja clicar e digitar automaticamente durante horas intermináveis. Você chegou até este ponto do livro; você sabe que, *obviamente*, deve haver outra maneira.

O formulário para esse projeto é um formulário do Google Docs que poderá ser encontrado em *http://nostarch.com/automatestuff*. Sua aparência é semelhante à da figura 18.4.

Figura 18.4 – O formulário usado nesse projeto.

De modo geral, eis o que o seu programa deverá fazer:

- Clicar no primeiro campo de texto do formulário.
- Percorrer o formulário digitando as informações em cada campo.
- Clicar no botão Submit (Submeter).
- Repetir o processo com o próximo conjunto de dados.

Isso significa que o seu código deverá fazer o seguinte:

- Chamar pyautogui.click() para clicar no formulário e no botão Submit.
- Chamar pyautogui.typewrite() para inserir texto nos campos.
- Tratar a exceção KeyboardInterrupt para que o usuário possa pressionar CTRL-C para sair.

Abra uma nova janela no editor de arquivo e salve esse arquivo como *formFiller.py*.

Passo 1: Identificar os passos

Antes de escrever o código, é preciso descobrir exatamente quais são as teclas e os cliques de mouse para preencher o formulário uma vez. O script

mouseNow.py poderá ajudar você a descobrir as coordenadas específicas do mouse. Você deverá conhecer as coordenadas somente do primeiro campo de texto. Após clicar no primeiro campo, você poderá simplesmente pressionar TAB para que o foco seja deslocado para o próximo campo. Isso evitará que seja necessário descobrir as coordenadas x e y em que um clique deverá ser feito para cada campo.

Eis os passos para inserir os dados no formulário:

1. Clique no campo Name (Nome). (Utilize *mouseNow.py* para determinar as coordenadas após ter maximizado a janela do navegador. No OS X, talvez seja necessário clicar duas vezes: uma para colocar o foco no navegador e outra para clicar no campo Name.)
2. Digite um nome (Name) e pressione TAB.
3. Digite o que a pessoa mais teme [Greatest Fear(s)] e pressione TAB.
4. Pressione a seta para baixo um número correto de vezes para selecionar a fonte de poder da magia (source of wizard powers): uma vez para *wand* (varinha), duas vezes para *amulet* (amuleto), três vezes para *crystal ball* (bola de cristal) e quatro vezes para *money* (dinheiro). Em seguida, pressione TAB. (Observe que, no OS X, você deverá pressionar a seta para baixo uma vez mais para cada opção. Em alguns navegadores, talvez seja necessário pressionar a tecla ENTER também.)
5. Pressione a seta para a direita para selecionar a resposta à pergunta sobre RoboCop. Pressione uma vez para *2*, duas vezes para *3*, três vezes para *4* ou quatro vezes para *5*, ou simplesmente pressione a barra de espaço para selecionar *1* (que estará em destaque por default). Em seguida, pressione TAB.
6. Digite um comentário adicional (Additional Comments) e pressione TAB.
7. Pressione a tecla ENTER para “clicar” no botão Submit.
8. Após submeter o formulário, o navegador conduzirá você a uma página em que será necessário clicar em um link para retornar à página do formulário.

Observe que, se esse programa for executado novamente mais tarde, talvez seja necessário atualizar as coordenadas dos cliques de mouse, pois a janela do navegador poderá ter mudado de posição. Para contornar esse problema,

sempre garanta que a janela do navegador esteja maximizada antes de determinar as coordenadas do primeiro campo do formulário. Além disso, navegadores diferentes em sistemas operacionais distintos poderão funcionar de modo um pouco diferente dos passos apresentados aqui, portanto verifique se essas combinações de teclas funcionam em seu computador antes de executar o seu programa.

Passo 2: Definir as coordenadas

Carregue o formulário de exemplo baixado (Figura 18.4) em um navegador e maximize a sua janela. Abra uma nova janela do Terminal ou de linha de comando para executar o script *mouseNow.py* e passe o mouse sobre o campo Name (Nome) para descobrir suas coordenadas x e y. Esses números serão atribuídos à variável nameField em seu programa. Além disso, encontre as coordenadas x e y e o valor da tupla RGB do botão Submit azul. Esses valores serão atribuídos às variáveis submitButton e submitButtonColor, respectivamente.

Em seguida, preencha alguns dados dummy no formulário e clique em Submit (Submeter). Você deverá ver a aparência da próxima página para que possa usar *mouseNow.py* e determinar as coordenadas do link *Submit another response* (Submeter outra resposta) nessa nova página.

Certifique-se de que seu código-fonte tenha a aparência a seguir e não se esqueça de substituir todos os valores em itálico pelas coordenadas que você determinar em seus próprios testes:

```
#! python3
# formFiller.py – Preenchimento automático do formulário.

import pyautogui, time

# Defina essas variáveis com as coordenadas corretas obtidas em seu computador.
nameField = (648, 319)
submitButton = (651, 817)
submitButtonColor = (75, 141, 249)
submitAnotherLink = (760, 224)

# TODO: Oferece ao usuário uma chance de encerrar o script.
```

```
# TODO: Espera até que a página do formulário tenha sido carregada.

# TODO: Preenche o campo Name.

# TODO: Preenche o campo Greatest Fear(s).

# TODO: Preenche o campo Source of Wizard Powers.

# TODO: Preenche o campo RoboCop.

# TODO: Preenche o campo Additional Comments.

# TODO: Clica em Submit.

# TODO: Espera até que a página do formulário tenha sido carregada.

# TODO: Clica no link Submit another response.
```

Agora você precisará dos dados que deseja realmente inserir nesse formulário. No mundo real, esses dados poderão estar em uma planilha, um arquivo com formato texto simples ou um site, e um código adicional será necessário para carregá-los no programa. Porém, nesse projeto, vamos deixar todos esses dados fixos no código em uma variável. Acrescente o seguinte em seu programa:

```
#! python3
# formFiller.py – Preenchimento automático do formulário.

--trecho removido--

formData = [{'name': 'Alice', 'fear': 'eavesdroppers', 'source': 'wand',
            'robocop': 4, 'comments': 'Tell Bob I said hi.'},
            {'name': 'Bob', 'fear': 'bees', 'source': 'amulet', 'robocop': 4,
            'comments': 'n/a'},
            {'name': 'Carol', 'fear': 'puppets', 'source': 'crystal ball',
            'robocop': 1, 'comments': 'Please take the puppets out of the
            break room.'},
            {'name': 'Alex Murphy', 'fear': 'ED-209', 'source': 'money',
            'robocop': 5, 'comments': 'Protect the innocent. Serve the public
```

```
 trust. Uphold the law.'},
]
```

--trecho removido--

A lista formData contém quatro dicionários para quatro nomes diferentes. Cada dicionário contém os nomes dos campos de texto como chaves e as respostas como valores. A última parte da configuração consiste em definir a variável PAUSE do PyAutoGUI para que haja uma pausa de meio segundo após cada chamada de função. Acrescente o seguinte em seu programa, após a instrução de atribuição de formData:

```
pyautogui.PAUSE = 0.5
```

Step 3: Começar a digitar os dados

Um loop for fará a iteração pelos dicionários da lista formData, passando os valores do dicionário às funções do PyAutoGUI que digitarão virtualmente os dados nos campos de texto.

Acrescente o código a seguir em seu programa:

```
#! python3
# formFiller.py – Preenchimento automático do formulário.

--trecho removido--

for person in formData:
    # Oferece ao usuário uma chance de encerrar o script.
    print('>>> 5 SECOND PAUSE TO LET USER PRESS CTRL-C <<<')
u   time.sleep(5)

    # Espera até que a página do formulário tenha sido carregada.
v   while not pyautogui.pixelMatchesColor(submitButton[0],
submitButton[1],  submitButtonColor):
        time.sleep(0.5)

--trecho removido--
```

Para proporcionar um pequeno recurso de segurança, o script faz uma pausa de cinco segundos u que oferece uma oportunidade ao usuário para teclar

CTRL-C (ou mover o cursor do mouse para o canto superior esquerdo da tela e gerar a exceção FailSafeException) a fim de encerrar o programa caso ele esteja fazendo algo inesperado. Então o programa espera até que a cor do botão Submit esteja visível v, permitindo que o programa saiba que a página do formulário foi carregada. Lembre-se de que você descobriu as informações de coordenada e de cor no passo 2 e as armazenou nas variáveis submitButton e submitButtonColor. Para usar pixelMatchesColor(), passe as coordenadas submitButton[0] e submitButton[1], além da cor em submitButtonColor.

Quando o código que espera a cor do botão Submit estiver visível, acrescente o seguinte:

```
#! python3
# formFiller.py – Preenchimento automático do formulário.

--trecho removido--

u   print('Entering %s info...' % (person['name']))
v   pyautogui.click(nameField[0], nameField[1])

    # Preenche o campo Name.
w   pyautogui.typewrite(person['name'] + '\t')

    # Preenche o campo Greatest Fear(s).
x   pyautogui.typewrite(person['fear'] + '\t')

--trecho removido--
```

Adicionamos uma chamada ocasional a print() para exibir o status do programa em sua janela do Terminal para que o usuário saiba o que está acontecendo u.

Como o programa sabe que o formulário foi carregado, é hora de chamar click() para clicar no campo Name (Nome) v e typewrite() para inserir a string em person['name'] w. O caractere '\t' é adicionado no final da string passada a typewrite() para simular o pressionamento de TAB, o que passará o foco do teclado para o próximo campo, que é Greatest Fear(s). Outra chamada a typewrite() fará a string person['fear'] ser digitada nesse campo, e TAB é usado para passar para o próximo campo do formulário x.

Passo 4: Tratar listas de seleção e botões de rádio

O menu suspenso da pergunta sobre "wizard powers" (poderes da magia) e os botões de rádio do campo RoboCop são mais complicados de tratar do que os campos de texto. Para clicar nessas opções com o mouse, você deverá descobrir as coordenadas x e y da cada opção possível. Em vez de fazer isso, será mais fácil usar as teclas de direção do teclado para fazer uma seleção.

Acrescente o seguinte em seu programa:

```
#! python3
# formFiller.py – Preenchimento automático do formulário.
--trecho removido--

    # Preenche o campo Source of Wizard Powers.
u   if person['source'] == 'wand':
v       pyautogui.typewrite(['down', '\t'])
    elif person['source'] == 'amulet':
        pyautogui.typewrite(['down', 'down', '\t'])
    elif person['source'] == 'crystal ball':
        pyautogui.typewrite(['down', 'down', 'down', '\t'])
    elif person['source'] == 'money':
        pyautogui.typewrite(['down', 'down', 'down', 'down', '\t'])

    # Preenche o campo RoboCop.
w   if person['robocop'] == 1:
x       pyautogui.typewrite([' ', '\t'])
    elif person['robocop'] == 2:
        pyautogui.typewrite(['right', '\t'])
    elif person['robocop'] == 3:
        pyautogui.typewrite(['right', 'right', '\t'])
    elif person['robocop'] == 4:
        pyautogui.typewrite(['right', 'right', 'right', '\t'])
    elif person['robocop'] == 5:
        pyautogui.typewrite(['right', 'right', 'right', 'right', '\t'])

--trecho removido--
```

Depois que o menu suspenso tiver o foco (lembre-se de que criamos o código para simular o pressionamento de TAB após o preenchimento do campo Greatest Fear(s)), usando a seta para baixo, passaremos para o

próximo item da lista de seleção. De acordo com o valor em person['source'], seu programa deverá enviar uma quantidade de pressionamentos da seta para baixo antes de enviar um TAB para o próximo campo. Se o valor da chave 'source' no dicionário desse usuário for 'wand' u, simularemos o pressionamento da seta para baixo uma vez (para selecionar *Wand*) e pressionaremos TAB v. Se o valor da chave 'source' for 'amulet', simularemos o pressionamento da seta para baixo duas vezes e pressionaremos TAB; o mesmo será feito sucessivamente para as demais respostas possíveis.

Os botões de rádio para a pergunta sobre RoboCop poderão ser selecionados com a seta para a direita – ou, se você quiser selecionar a primeira opção w, basta pressionar a barra de espaço x.

Passo 5: Submeter o formulário e esperar

Você pode preencher o campo Additional Comments (Comentários adicionais) com a função typewrite() ao passar person['comments'] como argumento. Um '\t' adicional pode ser digitado para passar o foco do teclado ao próximo campo ou ao botão Submit. Depois que o botão Submit tiver o foco, chamar pyautogui.press('enter') simulará o pressionamento da tecla ENTER e o formulário será submetido. Após a submissão do formulário, seu programa esperará cinco segundos para que a próxima página seja carregada.

Quando a nova página estiver carregada, ela terá um link *Submit another response* (Submeter outra resposta) que direcionará o navegador para uma nova página com um formulário em branco. Você armazenou as coordenadas desse link como uma tupla em submitAnotherLink no passo 2, portanto passe essas coordenadas a pyautogui.click() para clicar nesse link.

Com o novo formulário pronto, o loop for externo do script poderá passar para a próxima iteração e inserir as informações da próxima pessoa no formulário.

Complete o seu programa adicionando o código a seguir:

```
#! python3
# formFiller.py – Preenchimento automático do formulário.

--trecho removido--
```

```
# Preenche o campo Additional Comments.
pyautogui.typewrite(person['comments'] + '\t')

# Clica em Submit.
pyautogui.press('enter')

# Espera até que a página do formulário tenha sido carregada.
print('Clicked Submit.')
time.sleep(5)

# Clica no link Submit another response.
pyautogui.click(submitAnotherLink[0], submitAnotherLink[1])
```

Depois que o loop for principal tiver sido concluído, o programa terá inserido as informações de todas as pessoas. Nesse exemplo, há somente quatro pessoas cujas informações devem ser inseridas. Porém, se houvesse *4.000* pessoas, criar um programa para fazer isso faria você economizar bastante tempo e digitação!

Resumo

A automação de GUI com o módulo pyautogui permite interagir com aplicações em seu computador ao possibilitar o controle do mouse e do teclado. Embora essa abordagem seja flexível o suficiente para fazer tudo o que um usuário humano pode fazer, a desvantagem é que esses programas são bastante cegos em relação àquilo que clicam ou digitam. Ao escrever programas com automação de GUI, procure garantir que eles falharão rapidamente caso recebam instruções inadequadas. Falhar é irritante, porém é muito melhor do que o programa continuar errando.

Você pode mover o cursor do mouse pela tela e simular cliques de mouse, pressionamentos de teclas e atalhos de teclado com o PyAutoGUI. O módulo pyautogui também pode verificar as cores na tela, o que pode dar uma boa ideia do conteúdo da tela ao seu programa com automação de GUI para que ele saiba se está no caminho certo. Você pode até mesmo fornecer uma captura de tela ao PyAutoGUI para que ele descubra as coordenadas da área em que você deseja clicar.

Todos esses recursos do PyAutoGUI podem ser combinados para

automatizar qualquer tarefa repetitiva automática em seu computador. De fato, pode ser realmente hipnótico ver o cursor do mouse mover-se por conta própria e ver o texto aparecer na tela automaticamente. Por que não passar o tempo economizado sentado, vendo o seu programa fazer todo o trabalho para você? Há certa dose de satisfação proporcionada ao perceber que sua inteligência evitou a necessidade de realizar uma tarefa maçante.

Exercícios práticos

1. Como podemos disparar o recurso de falha com segurança do PyAutoGUI para interromper um programa?
2. Qual função retorna a resolução atual da tela?
3. Qual função retorna as coordenadas da posição atual do cursor do mouse?
4. Qual é a diferença entre pyautogui.moveTo() e pyautogui.moveRel()?
5. Quais funções podem ser usadas para arrastar o mouse?
6. Qual chamada de função digitará os caracteres de "Hello world!"?
7. Como podemos efetuar o pressionamento de teclas especiais, por exemplo, a seta para a esquerda do teclado?
8. Como podemos salvar o conteúdo atual da tela em um arquivo de imagem chamado *screenshot.png*?
9. Que código define uma pausa de dois segundos após toda chamada de função de PyAutoGUI?

Projetos práticos

Para exercitar, escreva programas que façam as tarefas a seguir.

Parecendo ocupado

Muitos programas de mensagens instantâneas determinam se você está ocioso ou distante de seu computador ao detectar uma ausência de movimentos de mouse durante um período de tempo – por exemplo, dez minutos. Talvez você queira sair de perto de sua escrivaninha por um instante, mas não queira que outras pessoas vejam que seu status no aplicativo de mensagens instantâneas mude para o modo ocioso. Crie um script que desloque um

pouco o cursor do mouse a cada dez segundos. Esse movimento deve ser pequeno o suficiente para que não atrapalhe se você precisar usar o seu computador enquanto o script estiver executando.

Bot para aplicativo de mensagens instantâneas

Google Talk, Skype, Yahoo Messenger, AIM e outros aplicativos de mensagens instantâneas normalmente utilizam protocolos proprietários que dificultam a criação de módulos Python para interagir com esses programas. Porém, mesmo esses protocolos proprietários não podem impedir você de escrever uma ferramenta com automação de GUI.

O aplicativo Google Talk tem uma barra de pesquisa que permite fornecer um nome de usuário de sua lista de amigos e abre uma janela de mensagens quando ENTER é pressionado. O foco do teclado será automaticamente passado para a nova janela. Outros aplicativos de mensagens instantâneas têm maneiras semelhantes de abrir novas janelas de mensagens. Crie um programa que envie automaticamente uma mensagem de notificação a um grupo selecionado de pessoas de sua lista de amigos. Seu programa talvez precise lidar com casos excepcionais, por exemplo, amigos que estejam offline, o fato de a janela do chat aparecer em coordenadas diferentes na tela ou o aparecimento de caixas de confirmação que interrompam a sua troca de mensagens. Seu programa deverá capturar telas para orientar a sua interação com a GUI e adotar maneiras de detectar quando suas teclas virtuais não estiverem sendo enviadas.

NOTA Você poderá criar algumas contas falsas de teste para não enviar spams acidentalmente aos seus amigos de verdade enquanto estiver desenvolvendo esse programa.

Tutorial para bot usado em jogo

Há um ótimo tutorial chamado "How to Build a Python Bot That Can Play Web Games" (Como criar um bot Python para jogar na web), acessível em *http://nostarch.com/automatestuff/*. Esse tutorial explica como criar um programa com automação de GUI em Python para usar um jogo Flash chamado Sushi Go Round. O jogo envolve clicar nos botões dos ingredientes corretos para atender aos pedidos de sushi dos clientes. Quanto mais rápido

atender aos pedidos sem cometer erros, mais pontos você ganhará. Essa é uma tarefa perfeitamente adequada para um programa com automação de GUI – e uma maneira de trapacear e conseguir uma pontuação elevada! O tutorial aborda vários dos mesmos assuntos discutidos neste capítulo, porém inclui também descrições dos recursos básicos de reconhecimento de imagens do PyAutoGUI.

APÊNDICE A

INSTALANDO MÓDULOS DE TERCEIROS

Além da biblioteca-padrão de módulos que acompanha o Python, outros desenvolvedores criaram seus próprios módulos para estender mais ainda as funcionalidades do Python. A maneira principal de instalar módulos de terceiros consiste em usar a ferramenta pip do Python. Essa ferramenta faz download e instala os módulos Python com segurança em seu computador a partir de *https://pypi.python.org/*, que é o site do Python Software Foundation. O PyPI – ou Python Package Index – é uma espécie de loja de aplicativos gratuitos para módulos Python.

Ferramenta pip

O arquivo executável da ferramenta pip se chama *pip* no Windows e *pip3* no OS X e no Linux. No Windows, o pip pode ser encontrado em *C:\Python34\Scripts\pip.exe*. No OS X, essa ferramenta está em */Library/Frameworks/Python.framework/Versions/3.4/bin/pip3*, e no Linux, em */usr/bin/pip3*.

Embora o pip seja instalado automaticamente com o Python 3.4 no Windows e no OS X, você deverá instalá-lo separadamente no Linux. Para instalar o pip3 no Ubuntu ou no Debian Linux, abra uma nova janela do Terminal e digite **sudo apt-get install python3-pip**. Para instalar o pip3 no Fedora Linux, digite **sudo yum install python3-pip** em uma janela do Terminal. Será necessário fornecer a senha de administrador de seu computador para instalar esse software.

Instalando módulos de terceiros

A ferramenta pip foi criada para ser executada a partir da linha de comando:

passe-lhe o comando install seguido do nome do módulo que você deseja instalar. Por exemplo, no Windows, digite pip install *NomeDoMódulo*, em que *NomeDoMódulo* é o nome do módulo. No OS X e no Linux, execute pip3 com o prefixo sudo para ter privilégios de administrador e instalar o módulo. Você deverá digitar sudo pip3 install *NomeDoMódulo*.

Se o módulo já estiver instalado, mas você quiser fazer um upgrade para a versão mais recente disponível no PyPI, execute pip install –U *NomeDoMódulo* (ou pip3 install –U *NomeDoMódulo* no OS X e no Linux).

Após instalar o módulo, você poderá testar se ele foi instalado com sucesso ao executar import *NomeDoMódulo* no shell interativo. Se nenhuma mensagem de erro for exibida, você saberá que o módulo foi instalado com sucesso.

Você pode instalar todos os módulos discutidos neste livro ao executar os comandos apresentados a seguir. (Lembre-se de substituir pip por pip3 se você estiver no OS X ou no Linux.)

- pip install send2trash
- pip install requests
- pip install beautifulsoup4
- pip install selenium
- pip install openpyxl
- pip install PyPDF2
- pip install python-docx (instale python-docx, e não docx)
- pip install imapclient
- pip install pyzmail
- pip install twilio
- pip install pillow
- pip install pyobjc-core (somente no OS X)
- pip install pyobjc (somente no OS X)
- pip install python3-xlib (somente no Linux)
- pip install pyautogui

NOTA Para usuários de OS X: o módulo pyobjc pode demorar vinte minutos ou mais para ser instalado, portanto não fique preocupado se demorar um pouco. Instale também o módulo pyobjc-core antes, o que reduzirá o tempo total de instalação.

APÊNDICE B

EXECUTANDO PROGRAMAS

Se você tiver um programa aberto no editor de arquivo do IDLE, executá-lo será simplesmente uma questão de pressionar F5 ou selecionar o item de menu Run4Run Module (Executar4Executar módulo). Essa é uma maneira fácil de executar programas enquanto você estiver escrevendo-os, porém abrir o IDLE para executar seus programas já concluídos pode ser inconveniente. Há maneiras mais apropriadas de executar scripts Python.

Linha shebang

A primeira linha de todos os seus programas Python deve ser uma linha *shebang*, que informa o computador que você quer que o Python execute esse programa. A linha começa com #!, porém o resto depende de seu sistema operacional.

- No Windows, a linha shebang é #! python3.
- No OS X, a linha shebang é #! /usr/bin/env python3.
- No Linux, a linha shebang é #! /usr/bin/python3.

Você poderá executar scripts Python a partir do IDLE sem a linha shebang, porém essa linha será necessária para executá-los a partir da linha de comando.

Executando programas Python no Windows

No Windows, o interpretador Python 3.4 está localizado em *C:\Python34\python.exe*. De modo alternativo, o programa *py.exe* conveniente lerá a linha shebang no início do código-fonte do arquivo *.py* e executará a versão apropriada do Python para esse script. O programa *py.exe* garantirá que o programa Python com a versão correta de Python seja executado se houver várias versões instaladas em seu computador.

Para que seja conveniente executar o seu programa Python, crie um arquivo batch *.bat* para executar o programa Python com *py.exe*. Para gerar um arquivo batch, crie um novo arquivo-texto que contenha uma única linha como esta:

```
@py.exe C:\path\to\your\pythonScript.py %*
```

Substitua esse path pelo path absoluto de seu próprio programa e salve esse arquivo com uma extensão *.bat* (por exemplo, *pythonScript.bat*). Esse arquivo batch evitará a necessidade de digitar o path absoluto completo do programa Python sempre que você quiser executá-lo. Recomendo que todos os seus arquivos batch e *.py* sejam colocados em uma única pasta, por exemplo, *C:\MyPythonScripts* ou *C:\Users\YourName\PythonScripts*.

A pasta *C:\MyPythonScripts* deve ser adicionada ao path do sistema no Windows para que você possa executar os arquivos batch dessa pasta no diálogo Run (Executar). Para isso, modifique a variável de ambiente PATH. Clique no botão Start (Iniciar) e digite Edit environment variables for your account (Editar as variáveis de ambiente para sua conta). Essa opção deverá ser preenchida automaticamente depois que você começar a digitá-la. A janela Environment Variables (Variáveis de Ambiente) apresentada deverá ter a aparência da figura B.1.

Em System variables (Variáveis do sistema), selecione a variável Path e clique em Edit (Editar). No campo de texto Value (Valor), acrescente ponto-e-vírgula, digite C:\MyPythonScripts e clique em OK. Agora você poderá executar qualquer script Python na pasta *C:\MyPythonScripts* simplesmente pressionando WIN-R e fornecendo o nome do script. Executar pythonScript, por exemplo, fará *pythonScript.bat* ser executado, que, por sua vez, evitará a necessidade de executar o comando completo py.exe C:\MyPythonScripts\pythonScript.py no diálogo Run (Executar).

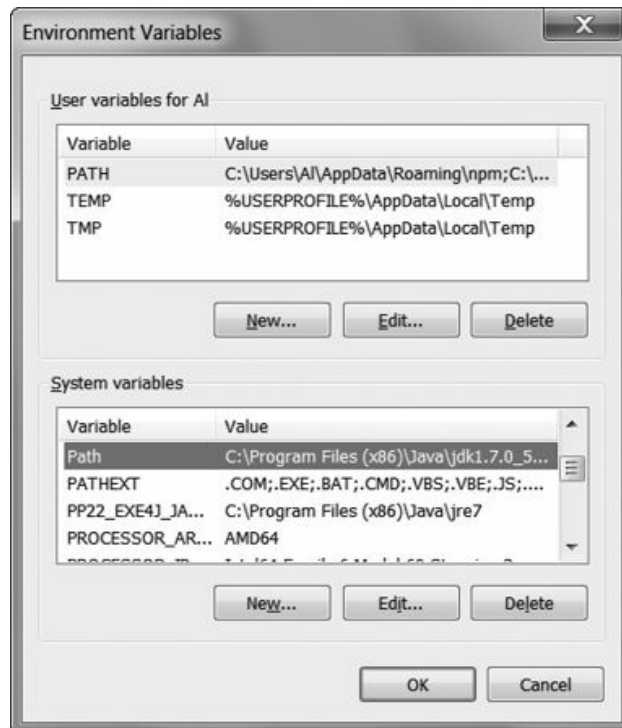



Figura B.1 – A janela Environment Variables (Variáveis de Ambiente) no Windows.

Executando programas Python no OS X e no Linux

No OS X, selecionar Applications4Utilities4Terminal (Aplicativos4Utilitários4Terminal) fará uma janela do *Terminal* ser apresentada. Uma janela do Terminal é uma maneira de especificar comandos em seu computador usando somente texto em vez de clicar em uma interface gráfica. Para que a janela do Terminal seja apresentada no Ubuntu Linux, pressione a tecla WIN (ou SUPER) para acessar o Dash e digite Terminal.

A janela do Terminal começará na pasta home de sua conta de usuário. Se o nome de meu usuário for *asweigart*, a pasta home será */Users/asweigart* no OS X e */home/asweigart* no Linux. O caractere til (~) é um atalho para a sua pasta home, portanto você poderá digitar cd ~ se quiser mudar para a sua pasta home. Você também pode usar o comando cd para alterar o diretório de trabalho atual para qualquer outro diretório. Tanto no OS X quanto no Linux, o comando pwd exibirá o diretório de trabalho atual.

Para executar seus programas Python, salve seu arquivo *.py* em sua pasta home. Em seguida, altere as permissões do arquivo *.py* para torná-lo executável por meio do comando chmod +x pythonScript.py. As permissões de

arquivo estão além do escopo deste livro, mas você deverá executar esse comando em seu arquivo Python se quiser executar o programa a partir da janela do Terminal. Depois que fizer isso, você poderá executar seu script sempre que quiser ao abrir uma janela do Terminal e digitar ./pythonScript.py. A linha shebang no início do script informará ao sistema operacional em que local o interpretador Python poderá ser encontrado.

Executando programas Python com as asserções desabilitadas

Você pode desabilitar as instruções assert em seus programas Python para ter uma pequena melhoria no desempenho. Ao executar o Python a partir do terminal, inclua a opção -O após python ou python3 e antes do nome do arquivo *.py*. Isso fará uma versão otimizada de seu programa ser executada, em que as verificações de asserção serão ignoradas.

APÊNDICE C

RESPOSTAS AOS EXERCÍCIOS PRÁTICOS

Este apêndice contém as respostas aos exercícios práticos no final de cada capítulo. Recomendo intensamente que você reserve tempo para trabalhar nesses problemas. A programação é mais do que memorizar a sintaxe e uma lista de nomes de funções. Assim como no aprendizado de uma língua estrangeira, quanto mais praticar, mais proveito você tirará dela. Além disso, há vários sites com problemas práticos de programação. Você poderá encontrar uma lista deles em *http://nostarch.com/automatestuff/*.

Capítulo 1

1. Os operadores são: +, -, * e /. Os valores são 'hello', -88.8 e 5.
2. A string é 'spam'; a variável é spam. As strings sempre começam e terminam com aspas.
3. Os três tipos de dados apresentados neste capítulo são: inteiros, números de ponto flutuante e strings.
4. Uma expressão é uma combinação de valores e de operadores. Todas as expressões são avaliadas como (isto é, reduzidas a) um único valor.
5. Uma expressão é avaliada como um único valor. Uma instrução não é.
6. A variável bacon estará definida com 20. A expressão bacon + 1 não atribui o valor novamente a bacon (para isso, seria necessária uma instrução de atribuição: bacon = bacon + 1).
7. Ambas as expressões são avaliadas como a string 'spamspamspam'.
8. Nomes de variáveis não podem começar com um número.
9. As funções int(), float() e str() serão avaliadas como as versões de inteiro, número de ponto flutuante e de string do valor passado a elas.
10. A expressão causa um erro porque 99 é um inteiro e somente strings podem ser concatenadas a outras strings com o operador +. A forma correta

é I have eaten ' + str(99) + ' burritos.'.

Capítulo 2

1. True e False, usando *T* e *F* maiúsculos, com o restante da palavra em letras minúsculas.
2. and, or e not
3. True and True é True.
   True and False é False.
   False and True é False.
   False and False é False.
   True or True é True.
   True or False é True.
   False or True é True.
   False or False é False.
   not True é False.
   not False é True.
4. False
   False
   True
   False
   False
   True
5. ==, !=, <, >, <= e >=.
6. == é o operador "igual a", que compara dois valores e é avaliado como um booleano, enquanto = é o operador de atribuição, que armazena um valor em uma variável.
7. Uma condição é uma expressão usada em uma instrução de controle de fluxo e é avaliada como um valor booleano.
8. Os três blocos são: tudo que está contido na instrução if e as linhas print('bacon') e print('ham').

```
print('eggs')
if spam > 5:
```

```
    print('bacon')
else:
    print('ham')
print('spam')
```

9. O código:

```
if spam == 1:
    print('Hello')
elif spam == 2:
    print('Howdy')
else:
    print('Greetings!')
```

10. Tecle CTRL-C para interromper um programa que estiver preso em um loop infinito.
11. A instrução break desviará a execução para fora do loop, imediatamente depois dele. A instrução continue desviará a execução para o início do loop.
12. Todas as chamadas fazem o mesmo. A chamada a range(10) varia de 0 a 10 (sem incluir esse valor), range(0, 10) diz explicitamente ao loop para iniciar em 0 e range(0, 10, 1) diz explicitamente ao loop para incrementar a variável de 1 a cada iteração.
13. O código:

```
for i in range(1, 11):
    print(i)
```

e:

```
i = 1
while i <= 10:
    print(i)
    i = i + 1
```

14. Essa função pode ser chamada com spam.bacon().

Capítulo 3

1. As funções reduzem a necessidade de códigos duplicados. Isso torna seus programas mais compactos, legíveis e fáceis de atualizar.
2. O código de uma função é executado quando a função é chamada, não quando ela é definida.
3. A instrução def define (ou seja, cria) uma função.
4. Uma função é constituída da instrução def e do código em sua cláusula def. Uma chamada de função é o que faz a execução do programa desviar-se para a função, e a chamada da função é avaliada com o valor de retorno da função.
5. Há um escopo global, e um escopo local é criado sempre que uma função é chamada.
6. Quando uma função retorna, o escopo local é destruído e todas as variáveis contidas nele serão esquecidas.
7. Um valor de retorno é o valor com o qual uma chamada de função é avaliada. Como qualquer valor, um valor de retorno pode ser usado como parte de uma expressão.
8. Se não houver nenhuma instrução de retorno em uma função, seu valor de retorno será None.
9. Uma instrução global forçará uma variável em uma função a se referir à variável global.
10. O tipo de dado de None é NoneType.
11. Essa instrução import importa um módulo chamado areallyourpetsnamederic. (A propósito, esse não é um nome de módulo verdadeiro em Python.)
12. Essa função pode ser chamada com spam.bacon().
13. Insira a linha de código que possa provocar um erro em uma cláusula try.
14. O código que pode provocar um erro em potencial é inserido em uma cláusula try. O código executado se um erro ocorrer é inserido na cláusula except.

Capítulo 4

1. É o valor de lista vazia, que é um valor de lista que não contém nenhum item. É semelhante ao modo como '' representa o valor de string vazia.
2. spam[2] = 'hello' (Observe que o terceiro valor em uma lista está no índice 2, pois o primeiro índice é 0.)
3. 'd' (Observe que '3' * 2 é a string '33', que é passada para int() antes de ser dividida por 11. Esse valor será avaliado como 3. As expressões podem ser usadas em qualquer lugar em que os valores são utilizados.)
4. 'd' (Índices negativos são contabilizados a partir do final.)
5. ['a', 'b']
6. 1
7. [3.14, 'cat', 11, 'cat', True, 99]
8. [3.14, 11, 'cat', True]
9. O operador para concatenação de lista é +, enquanto o operador de repetição é *. (É o mesmo para strings.)
10. Enquanto append() adicionará valores somente no final de uma lista, insert() poderá adicioná-los em qualquer ponto.
11. A instrução del e o método de lista remove() são duas maneiras de remover valores de uma lista.
12. Tanto as listas quanto as strings podem ser passadas para len(), têm índices e slices, podem ser usadas em loops for, podem ser concatenadas ou repetidas e podem ser utilizadas com os operadores in e not in.
13. As listas são mutáveis; elas podem ter valores adicionados, removidos ou alterados. As tuplas são imutáveis; elas não podem ser alteradas. Além disso, as tuplas são escritas com parênteses, ou seja, (e), enquanto as listas utilizam colchetes, ou seja, [e].
14. (42,) (A vírgula final é obrigatória.)
15. Usando as funções tuple() e list(), respectivamente.
16. Elas contêm referências a valores de lista.
17. A função copy.copy() fará uma cópia rasa (shallow copy) de uma lista, enquanto a função copy.deepcopy() fará uma cópia profunda (deep copy). Isso quer dizer que somente copy.deepcopy() duplicará qualquer lista que estiver na lista.

Capítulo 5

1. Duas chaves: {}
2. {'foo': 42}
3. Os itens armazenados em um dicionário não estão ordenados, enquanto os itens em uma lista estão.
4. Você obterá um erro KeyError.
5. Não há nenhuma diferença. O operador in verifica se um valor existe como chave no dicionário.
6. 'cat' in spam verifica se há uma chave 'cat' no dicionário, enquanto 'cat' in spam.values() verifica se há um valor 'cat' para uma das chaves em spam.
7. spam.setdefault('color', 'black')
8. pprint.pprint()

Capítulo 6

1. Os caracteres de escape representam caracteres em valores de string que, do contrário, seriam difíceis ou impossíveis de digitar no código.
2. \n é uma quebra de linha; \t é uma tabulação.
3. O caractere de escape \\ representa um caractere de barra invertida.
4. O caractere de aspas simples em Howl's não representa um problema porque você utilizou aspas duplas para marcar o início e o fim da string.
5. As strings de múltiplas linhas permitem usar quebras de linha em strings sem o caractere de escape \n.
6. As expressões são avaliadas como:
   - 'e'
   - 'Hello'
   - 'Hello'
   - 'lo world!
7. As expressões são avaliadas como:
   - 'HELLO'
   - True

- 'hello'

8. As expressões são avaliadas como:
   - ['Remember,', 'remember,', 'the', 'fifth', 'of', 'November.']
   - 'There-can-be-only-one.'
9. Os métodos de string rjust(), ljust() e center(), respectivamente.
10. Os métodos lstrip() e rstrip() removem caracteres de espaços em branco das extremidades esquerda e direita de uma string, respectivamente.

Capítulo 7

1. A função re.compile() retorna objetos Regex.
2. Strings puras são usadas para que as barras invertidas não precisem ser escapadas.
3. O método search() retorna objetos Match.
4. O método group() retorna strings com o texto correspondente.
5. O grupo 0 é a correspondência completa, o grupo 1 inclui o primeiro conjunto de parênteses e o grupo 2 inclui o segundo grupo de parênteses.
6. Os pontos e os parênteses podem ser escapados com uma barra invertida: \., \(e \).
7. Se a regex não tiver nenhum grupo, uma lista de strings será retornada. Se a regex tiver grupos, uma lista de tuplas de strings será retornada.
8. O caractere | quer dizer uma correspondência de "um ou outro" entre dois grupos.
9. O caractere ? pode significar "corresponda a zero ou a uma ocorrência do grupo anterior" ou pode ser usado para indicar uma correspondência nongreedy.
10. + corresponde a um ou mais; * corresponde a zero ou mais.
11. {3} corresponde a exatamente três ocorrências do grupo anterior; {3,5} corresponde a três até cinco instâncias.
12. As classes abreviadas de caracteres \d, \w e \s correspondem a um dígito, um caractere de palavra ou um caractere de espaço, respectivamente.

13. As classes abreviadas de caracteres \D, \W e \S correspondem a um único caractere que não seja um dígito, não seja um caractere de palavra ou não seja um caractere de espaço, respectivamente.
14. Passar re.I ou re.IGNORECASE como segundo argumento de re.compile() fará a correspondência ignorar a diferença entre letras maiúsculas e minúsculas.
15. O caractere . normalmente corresponde a qualquer caractere, exceto o caractere de quebra de linha. Se re.DOTALL for passado como segundo argumento de re.compile(), o ponto também corresponderá aos caracteres de quebra de linha.
16. .* realiza uma correspondência greedy e .*? realiza uma correspondência nongreedy.
17. [0-9a-z] ou [a-z0-9]
'18. X drummers, X pipers, five rings, X hens'
19. O argumento re.VERBOSE permite adicionar espaços em branco e comentários à string passada para re.compile().
20. re.compile(r'^\d{1,3}(,\d{3})*$') criará essa regex, porém outras strings de regex podem gerar uma expressão regular semelhante.
21. re.compile(r'[A-Z][a-z]*\sNakamoto')
22. re.compile(r'(Alice|Bob|Carol)\s(eats|pets|throws)\s(apples|cats|baseballs)\.', re.IGNORECASE)

Capítulo 8

1. Paths relativos são relativos ao diretório de trabalho atual.
2. Os paths absolutos começam com a pasta-raiz, por exemplo, */* ou *C:*.
3. A função os.getcwd() retorna o diretório de trabalho atual. A função os.chdir() *altera* o diretório de trabalho atual.
4. A pasta . é a pasta atual e .. é a pasta-pai.
5. *C:\bacon\eggs* é o nome do diretório, enquanto *spam.txt* é o nome base.
6. São: a string 'r' para o modo de leitura, 'w' para o modo de escrita e 'a' para o modo de adição.

7. Um arquivo existente aberto em modo de escrita será apagado e totalmente sobrescrito.
8. O método read() retorna todo o conteúdo do arquivo como um único valor de string. O método readlines() retorna uma lista de strings em que cada string é uma linha do conteúdo do arquivo.
9. Um valor de shelf lembra um valor de dicionário; ele tem chaves e valores, juntamente com métodos keys() e values() que funcionam de modo semelhante aos métodos de dicionário com os mesmos nomes.

Capítulo 9

1. A função shutil.copy() copia um único arquivo, enquanto shutil.copytree() copia uma pasta toda, juntamente com todo o seu conteúdo.
2. A função shutil.move() é usada para renomear arquivos, assim como para movê-los.
3. As funções em send2trash movem um arquivo ou uma pasta para a lixeira, enquanto as funções em shutil apagam os arquivos e as pastas permanentemente.
4. A função zipfile.ZipFile() é equivalente à função open(); o primeiro argumento é o nome do arquivo e o segundo argumento é o modo com o qual o arquivo ZIP será aberto (modo de leitura, de escrita ou de adição).

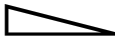
Capítulo 10

1. assert(spam >= 10, 'The spam variable is less than 10.')
2. assert(eggs.lower() != bacon.lower(), 'The eggs and bacon variables are the same!') ou assert(eggs.upper() != bacon.upper(), 'The eggs and bacon variables are the same!')
3. assert(False, 'This assertion always triggers.')
4. Para que logging.debug() possa ser chamado, as duas linhas a seguir devem estar no início de seu programa:

```
import logging
logging.basicConfig(level=logging.DEBUG, format=' %(asctime)s -  %(levelname)s
-  %(message)s')
```

5. Para poder enviar mensagens de logging a um arquivo chamado *programLog.txt* com logging.debug(), as duas linhas a seguir devem estar no início de seu programa:

```
import logging
>>> logging.basicConfig(filename='programLog.txt', level=logging.DEBUG, format='
%(asctime)s -  %(levelname)s -  %(message)s')
```

6. DEBUG, INFO, WARNING, ERROR e CRITICAL
7. logging.disable(logging.CRITICAL)
8. Podemos desabilitar as mensagens de logging sem remover as chamadas às funções de logging. Podemos desabilitar seletivamente as mensagens de logging de níveis mais baixos. Podemos criar mensagens de logging. As mensagens de logging podem disponibilizar um timestamp.
9. O botão Step fará o debugger desviar-se para dentro de uma chamada de função. O botão Over executará rapidamente a chamada de função sem entrar nela. O botão Out executará rapidamente o restante do código até sair da função em que estiver no momento.
10. Após clicar em Go, o debugger será interrompido quando alcançar o final do programa ou uma linha que contenha um breakpoint.
11. Um breakpoint é uma configuração em uma linha de código que faz o debugger fazer uma pausa quando a execução do programa alcançar essa linha.
12. Para definir um breakpoint no IDLE, clique com o botão direito do mouse na linha e selecione Set Breakpoint (Definir breakpoint) no menu de contexto.

Capítulo 11

1. O módulo webbrowser tem um método open() que apenas iniciará um navegador web em um URL específico. O módulo requests pode fazer download de arquivos e de páginas da Web. O módulo BeautifulSoup faz parse de HTML. Por fim, o módulo selenium pode iniciar e controlar um navegador.

2. A função requests.get() retorna um objeto Response; seu atributo text contém os dados baixados na forma de uma string.
3. O método raise_for_status() gera uma exceção se o download teve problemas e não faz nada se foi bem-sucedido.
4. O atributo status_code do objeto Response contém o código de status HTTP.
5. Após abrir o novo arquivo em seu computador em modo 'wb' de “escrita binária”, utilize um loop for que faça uma iteração pelo método iter_content() do objeto Response e grave porções de dados no arquivo. Aqui está um exemplo:

```
saveFile = open('filename.html', 'wb')
for chunk in res.iter_content(100000):
    saveFile.write(chunk)
```

6. F12 faz as ferramentas de desenvolvedor serem apresentadas no Chrome. Pressionar CTRL-SHIFT-C (no Windows e no Linux) ou -OPTION-C (no OS X) apresenta as ferramentas de desenvolvedor no Firefox.
7. Clique com o botão direito do mouse no elemento da página e selecione Inspect Element (Inspecionar elemento) no menu.
8. '#main'
9. '.highlight'
10. 'div div'
11. 'button[value="favorite"]'
12. spam.getText()
13. linkElem.attrs
14. O módulo selenium é importado com from selenium import webdriver.
15. Os métodos find_element_* retornam o primeiro elemento correspondente como um objeto WebElement. Os métodos find_elements_* retornam uma lista com todos os elementos correspondentes na forma de objetos WebElement.
16. Os métodos click() e send_keys() simulam cliques de mouse e pressionamentos de teclas, respectivamente.
17. Chamar o método submit() em qualquer elemento de um formulário submete o formulário.

18. Os métodos forward(), back() e refresh() do objeto WebDriver simulam esses botões do navegador.

Capítulo 12

1. A função openpyxl.load_workbook() retorna um objeto Workbook.
2. O método get_sheet_names() retorna um objeto Worksheet.
3. Chame wb.get_sheet_by_name('Sheet1').
4. Chame wb.get_active_sheet().
5. Utilize sheet['C5'].value ou sheet.cell(row=5, column=3).value.
6. Utilize sheet['C5'] = 'Hello' ou sheet.cell(row=5, column=3).value = 'Hello'.
7. Utilize cell.row e cell.column.
8. Elas retornam os maiores valores de coluna e de linha da planilha, respectivamente, como valores inteiros.
9. openpyxl.cell.column_index_from_string('M')
10. openpyxl.cell.get_column_letter(14)
11. Utilize sheet['A1':'F1'].
12. Utilize wb.save('example.xlsx').
13. Uma fórmula é definida da mesma maneira que qualquer valor. Defina o atributo value da célula com uma string contendo o texto da fórmula. Lembre-se de que as fórmulas começam com o sinal =.
14. Ao chamar load_workbook(), passe True para o argumento nomeado data_only.
15. Utilize sheet.row_dimensions[5].height = 100.
16. Utilize sheet.column_dimensions['C'].hidden = True.
17. O OpenPyXL 2.0.5 não carrega painéis congelados, não exibe títulos, imagens ou gráficos.
18. Os painéis congelados são linhas e colunas que sempre aparecerão na tela. São convenientes para cabeçalhos.
19. openpyxl.charts.Reference(), openpyxl.charts.Series(), openpyxl.charts.BarChart(), chartObj.append(seriesObj) e add_chart().

Capítulo 13

1. Um objeto File retornado por open().
2. Em modo de leitura binária ('rb') para PdfFileReader() e em modo de escrita binária ('wb') para PdfFileWriter().
3. Chamar getPage(4) retornará um objeto Page para a página cinco, pois a página zero é a primeira.
4. A variável numPages armazena um inteiro com o número de páginas do objeto PdfFileReader.
5. Devemos chamar decrypt('swordfish').
6. Os métodos rotateClockwise() e rotateCounterClockwise(). Os graus para rotação são passados como um argumento inteiro.
7. docx.Document('demo.docx')
8. Um documento contém vários parágrafos. Um parágrafo começa em uma nova linha e contém vários runs. Os runs são grupos contíguos de caracteres em um parágrafo.
9. Utilize doc.paragraphs.
10. Um objeto Run contém essas variáveis (e *não* um objeto Paragraph).
11. True sempre deixa o objeto Run em negrito e False sempre remove o negrito, independentemente da configuração de negrito do estilo. None fará o objeto Run simplesmente usar a configuração de negrito do estilo.
12. Chame a função docx.Document().
13. Utilize doc.add_paragraph('Hello there!').
14. Os inteiros 0, 1, 2, 3 e 4.

Capítulo 14

1. Em Excel, as planilhas podem ter valores com tipos de dados que não sejam strings, as células podem ter configurações de fonte, tamanho ou cores diferentes, as células podem ter larguras e alturas variadas, células adjacentes podem ser mescladas, e você pode incluir imagens e gráficos.
2. Passe um objeto File obtido a partir de uma chamada a open().

3. Os objetos File devem ser abertos em modo de leitura binária ('rb') para objetos Reader e em modo de escrita binária ('wb') para objetos Writer.
4. O método writerow().
5. O argumento delimiter altera a string usada para separar células em uma linha. O argumento lineterminator altera a string usada para separar linhas.
6. json.loads()
7. json.dumps()

Capítulo 15

1. É um instante de referência que muitos programas que manipulam data e hora utilizam. Esse instante corresponde a 1 de janeiro de 1970, UTC.
2. time.time()
3. Utilize time.sleep(5).
4. Essa função retorna o inteiro mais próximo do argumento passado. Por exemplo, round(2.4) retorna 2.
5. Um objeto datetime representa um instante específico no tempo. Um objeto timedelta representa uma duração.
6. threadObj = threading.Thread(target=spam)

threadObj.start()

7. Certifique-se de que o código que executar em uma thread não leia nem escreva nas mesmas variáveis que o código que estiver executando em outra thread.
8. subprocess.Popen('c:\\Windows\\System32\\calc.exe')

Capítulo 16

1. SMTP e IMAP, respectivamente.
2. smtplib.SMTP(), smtpObj.ehlo(), smptObj.starttls() e smtpObj.login().
3. imapclient.IMAPClient() e imapObj.login().
4. Uma lista de strings de palavras-chaves IMAP, por exemplo, 'BEFORE <date>', 'FROM <string>' ou 'SEEN'.

5. Atribua um valor inteiro maior à variável imaplib._MAXLINE, por exemplo, 10000000.
6. O módulo pyzmail lê os emails baixados.
7. Serão necessários: o número SID da conta Twilio, o número do token de autenticação e o seu número de telefone no Twilio.

Capítulo 17

1. Um valor RGBA é uma tupla de quatro inteiros, cada qual variando de 0 a 255. Os quatro inteiros correspondem à quantidade de vermelho (red), verde (green), azul (blue) e alpha (transparência) da cor.
2. Uma chamada de função a ImageColor.getcolor('CornflowerBlue', 'RGBA') retornará (100, 149, 237, 255), que é o valor RGBA dessa cor.
3. Uma tupla de caixa é um valor de tupla com quatro inteiros: a coordenada x da borda esquerda, a coordenada y da borda superior, a largura e a altura, respectivamente.
4. Utilize Image.open('zophie.png').
5. imageObj.size é uma tupla de dois inteiros com a largura e a altura.
6. imageObj.crop((0, 50, 50, 50)). Observe que devemos passar uma tupla de caixa para crop(), e não quatro argumentos inteiros separados.
7. Chame o método imageObj.save('new_filename.png') do objeto Image.
8. O módulo ImageDraw contém o código para desenhar em imagens.
9. Os objetos ImageDraw têm métodos para desenhar formas, como point(), line() ou rectangle(). Eles serão retornados se passarmos o objeto Image à função ImageDraw.Draw().

Capítulo 18

1. Mova o mouse para o canto superior esquerdo da tela, ou seja, para as coordenadas (0, 0).
2. pyautogui.size() retorna uma tupla com dois inteiros correspondentes à largura e à altura da tela.

3. pyautogui.position() retorna uma tupla com dois inteiros para as coordenadas x e y do cursor do mouse.
4. A função moveTo() move o mouse para coordenadas absolutas na tela, enquanto a função moveRel() move o mouse em relação à sua posição atual.
5. pyautogui.dragTo() e pyautogui.dragRel().
6. Utilize pyautogui.typewrite('Hello world!').
7. Passe uma lista de strings de teclas para pyautogui.typewrite() (por exemplo, 'left') ou passe uma única string de tecla para pyautogui.press().
8. Utilize pyautogui.screenshot('screenshot.png').
9. Utilize pyautogui.PAUSE = 2.

Black Hat Python
Programação Python para hackers e pentesters
Justin Seitz
novatec
Apresentação de Charlie Miller

JOVEM E
BEM-SUCEDIDO
Um guia para a realização
profissional e financeira
TRABALHO
DINHEIRO
ESTUDOS
novatec
Juliano Niederauer

Jovem e Bem-sucedido

Niederauer, Juliano

9788575225325

192 páginas

Jovem e Bem-sucedido é um verdadeiro guia para quem deseja alcançar a realização profissional e a financeira o mais rápido possível. Repleto de dicas e histórias interessantes vivenciadas pelo autor, o livro desmistifica uma série de crenças relativas aos estudos, ao trabalho e ao dinheiro.

Tem como objetivo orientar o leitor a planejar sua vida desde cedo, possibilitando que se torne bem-sucedido em pouco tempo e consiga manter essa realização no decorrer dos anos. As três perspectivas abordadas são:

ESTUDOS: mostra que os estudos vão muito além da escola ou faculdade. Aborda as melhores práticas de estudo e a aquisição dos conhecimentos ideais e nos momentos certos.

TRABALHO: explica como você pode se tornar um profissional moderno, identificando oportunidades e aumentando cada vez mais suas fontes de renda. Fornece ainda dicas valiosas para desenvolver as habilidades mais valorizadas no mercado de trabalho.

DINHEIRO: explica como assumir o controle de suas finanças, para, então,

começar a investir e multiplicar seu patrimônio. Apresenta estratégias de investimentos de acordo com o momento de vida de cada um, abordando as vantagens e desvantagens de cada tipo de investimento.

Jovem e Bem-sucedido apresenta ideias que o acompanharão a vida toda, realizando importantes mudanças no modo como você planeja estudar, trabalhar e lidar com o dinheiro.

WIRESHARK

PARA PROFISSIONAIS DE SEGURANÇA

Usando Wireshark e o Metasploit Framework

WILEY
novatec

Jessey Bullock
Jeff T. Parker

mostram como o Wireshark é usado em uma rede de verdade, com o ambiente virtual Docker disponibilizado; além disso, princípios básicos de rede e de segurança são explicados em detalhes para ajudar você a entender o porquê, juntamente com o como. Ao usar a distribuição Kali Linux para testes de invasão, em conjunto com o laboratório virtual e as capturas de rede disponibilizadas, você poderá acompanhar os diversos exemplos ou até mesmo começar a pôr em prática imediatamente o seu conhecimento em um ambiente de rede seguro. A experiência prática torna-se mais valiosa ainda pela ênfase em uma aplicação coesa, ajudando você a explorar vulnerabilidades e a expandir todas as funcionalidades do Wireshark, estendendo-as ou integrando-as com outras ferramentas de segurança.

Definindo Escopo em
Projetos de Software
novatec
Carlos Alberto Debastiani

Definindo Escopo em Projetos de Software

Debastiani, Carlos Alberto

9788575224960

144 páginas

Compre agora e leia

Definindo Escopo em Projetos de Software é uma obra que pretende tratar, de forma clara e direta, a definição de escopo como o fator mais influente no sucesso dos projetos de desenvolvimento de sistemas, uma vez que exerce forte impacto sobre seus custos. Abrange diversas áreas do conhecimento ligadas ao tema, abordando desde questões teóricas como a normatização e a definição das características de engenharia de software, até questões práticas como métodos para coleta de requisitos e ferramentas para desenho e projeto de soluções sistêmicas.

Utilizando uma linguagem acessível, diversas ilustrações e citações de casos vividos em sua própria experiência profissional, o autor explora, de forma abrangente, os detalhes que envolvem a definição de escopo, desde a identificação das melhores fontes de informação e dos envolvidos na tomada de decisão, até as técnicas e ferramentas usadas no levantamento de requisitos, no projeto da solução e nos testes de aplicação.

Compre agora e leia

Manual do Futuro
Redator
Técnicas e causos para quem
quer se aventurar na profissão
novatec
Sérgio Calderaro

Manual do Futuro Redator

Calderaro, Sérgio

9788575224908

120 páginas

Você estuda ou está pensando em estudar Publicidade, Jornalismo ou Letras? Gosta de ler e escrever e quer dicas de quem já passou por poucas e boas em agências de publicidade, redações de jornal e editoras? Quer conhecer causos curiosos de um profissional do texto que já deu aulas em universidades do Brasil e da Europa e trabalhou como assessor de Imprensa e Divulgação em uma das maiores embaixadas brasileiras do exterior?

O Manual do futuro redator traz tudo isso e muito mais. Em linguagem ágil e envolvente, mescla orientações técnicas a saborosas histórias do dia a dia de um profissional com duas décadas e meia de ofício. Esta obra inédita em sua abordagem pretende fazer com que você saiba onde está se metendo antes de decidir seu caminho. Daí pra frente, a decisão será sua. Vai encarar?

Compre agora e leia